



图机器学习与图神经网络

Stanford CS224W 课程学习笔记

作者：CC98@ 小角龙

时间：2021.06-2021.07

版本：1.0

自定义：信息



前言

Stanford CS224W Machine Learning with Graphs 课程学习笔记，内容参考课程 slides 整理而成，仅供交流使用。

Zhang Each
June 29, 2021

目录

1	图机器学习导论	1
1.1	图机器学习研究的问题	1
1.2	图的表示方式	1
2	传统的图机器学习方法	2
2.1	节点层面的任务	2
2.1.1	节点的 Centrality	2
2.1.1.1	特征向量中心	2
2.1.1.2	中间性中心 Betweenness Centrality	2
2.1.1.3	邻近中心 Closeness Centrality	3
2.1.2	聚类系数	3
2.1.3	Graphlets	3
2.2	边层面的任务	4
2.2.1	基于距离的特征	4
2.2.2	Local Neighborhood Overlap	4
2.2.3	Global Neighborhood Overlap	4
2.3	图层面的任务	5
2.3.1	BoW	5
2.3.2	Graphlet 特征	5
3	节点嵌入	7
3.1	编码器和解码器	7
3.1.1	关键组件	7
3.1.2	浅编码 Shallow Encoder	7
3.2	随机游走 Random Walk	8
3.2.1	什么是随机游走	8
3.2.2	为什么需要随机游走	8
3.2.3	随机游走的优化	8
3.3	node2vec	9
3.3.1	有偏见的随机游走	9
3.3.2	node2vec 算法框架	10
3.4	图嵌入	10
4	链接分析: PageRank 算法	12
4.1	PageRank 算法细节	12
4.1.1	节点的权重	12
4.1.2	图中的随机游走	12
4.2	求解 PageRank	12
4.3	个性化的 PageRank	13

5 消息传递和节点分类	14
5.1 关系分类	14
5.2 迭代分类	14
5.3 协作分类：信任传播机制	15
5.3.1 一个简单的消息传播的例子	15
5.3.2 循环消息传递算法	15
6 图神经网络 GNN	17
6.1 深度学习基础	17
6.2 图深度学习	17
6.2.1 问题的定义与 naive approach	17
6.2.2 图卷积网络的结构	18
6.2.3 如何训练 GNN	19
6.3 经典的 GNN 模型	19
6.3.1 GNN 的层级结构	20
6.3.1.1 GCN	20
6.3.1.2 GraphSAGE	20
6.3.1.3 GAT 与注意力机制	21
6.3.2 通用的 GNN 层设计	22
6.3.3 GNN 的层级架构	23
6.3.3.1 过平滑问题和接收域	23
6.3.3.2 解决过平滑问题	24
6.3.3.3 Skip Connection	24
6.3.4 GNN 对图的预处理	25
7 GNN 的设计、训练和应用	26
7.1 GNN 面对的任务	26
7.1.1 节点级别的任务	26
7.1.2 边级别的任务	26
7.1.3 图级别的任务	27
7.2 监督学习和无监督学习	27
7.3 损失函数和评价标准的选取	28
7.3.1 损失函数	28
7.3.2 评估标准	28
7.4 数据集的划分	28
8 图神经网络的表示能力	29
8.1 局部结构和计算图	29
8.2 GNN 的设计	30
8.2.1 经典架构	30
8.2.2 MLP 和 GIN	30
9 知识图谱	31
9.1 关系型图卷积网络 RGCN	31
9.2 知识图谱	31

9.2.1 知识图谱中的关系模式	32
9.2.2 TransE	32
9.2.3 TransR	32
9.2.4 Bilinear Modeling	32
9.2.5 DisMult	33
9.2.6 ComplEx	33
9.2.7 知识图谱嵌入算法总结	33
10 知识图谱推理	34
10.1 Reasoning 和 query	34
10.2 query 的形式化表述	35
10.2.1 one-hop query	35
10.2.2 path query	35
10.3 知识图谱推理和问答	35
10.4 Query2Box	36
10.4.1 投影操作	37
10.4.2 取交集 (Intersection) 操作	37
10.4.3 Query2Box 详解	37
10.4.4 打分函数	38
10.4.5 Union 操作的扩展	38
10.4.6 Query2Box 的训练	39
10.4.7 query 的生成	40
11 频繁子图挖掘 Frequent Subgraph Mining	41
11.1 子图	41
11.1.1 图的同构	41
11.1.2 网络主题 Network Motifs	41
11.1.3 子图的频率	42
11.2 Motif 的重要性	42
11.2.1 随机图	42
11.2.2 重要性计算的过程	43
11.2.3 Z-Score	43
11.3 GNN 与子图匹配	43
11.3.1 节点锚定	43
11.3.2 有序嵌入空间 Order Embedding Space	44
11.3.3 模型的构建和训练	45
11.3.4 频繁子图挖掘	45
11.3.5 使用 GNN 进行频繁子图挖掘	45
11.3.6 SPMiner 算法	46
12 社区检测	48
12.1 Edge Overlap	49
12.2 网络社区检测	49
12.2.1 Modularity 模块度	49
12.3 社区检测算法	50

13 图生成模型	51
13.1 传统的图生成模型	51
13.1.1 真实世界中图的性质	51
13.1.1.1 鲁棒性的衡量: Expansion	51
13.1.1.2 随机图模型中存在的问题	51
13.1.2 Small-World 图模型	51
13.1.3 Knnonecker 模型	52
13.2 深度图生成模型	53
13.2.1 机器学习方法	53
13.2.2 GraphRNN	53
13.2.2.1 模型简介	53
13.2.2.2 模型的训练和测试	54
13.2.2.3 存在的问题	55
13.2.3 评估生成的图	55
14 GNN 的前沿话题	56
14.1 GNN 的局限性	56
14.2 Position-aware GNN	56
14.3 Identity-aware GNN	57
14.4 GNN 的鲁棒性	58
14.4.1 问题的描述	59
14.4.2 问题的形式化	59
14.4.3 实验结果和结论	59

第 1 章 图机器学习导论

1.1 图机器学习研究的问题

图结构和网络结构是非常常见的数据结构，常见的事件图、计算机网络、事务网、社交网络和人际关系网等等都是图状结构，

- 网络也叫做自然图 Natural Graphs
- 图是对于一些关系的表示，可以表示信息和知识的联系，软件程序也可以表示为一张图
- 很多时候网络和图的区别是非常模糊的，通常可以将它们混为一谈，而该课程要研究的也就是如何利用图中的关系来进行更好的预测
- 相比于图像和文本，图结构的大小比较随意，没有固定的节点顺序，并且通常是动态的

图学习的任务可以有 node level, edge level, community level 和 graph-level 多个层次，常见的有：

- 节点分类：比如预测线上用户的类型
- 链接预测：预测两个节点之间是否缺少了链接，比如知识图谱补全
- 图分类：比如化学中的分子性质预测
- 聚类：检测节点是否构成一个类别，比如社交圈检测
- 其他的还有图生成，图进化等等

1.2 图的表示方式

常见的表示方式使用节点集合 N 和边集合 E 构成一个图结构 $G(N, E)$

- 图可以分为有向图和无向图，区别就在于边是否具有方向性，节点的 degree 表示包含这个节点的边的数量，而在有向图中又可以具体分为出度和入度
- 二分图是一种特殊的图结构，可以将图中的节点分成两个不相交的集合 U 和 V ，使得图中的任意一条边的两个顶点分别属于 U 和 V

图也可以用邻接矩阵来表示，其中无向图的邻接矩阵是一个对称矩阵，通常邻接矩阵都比较稀疏，网络结构通常都是比较稀疏的图，也可以用接邻表来表示，这种表示方式只保留了有边的节点的情况，而舍去了稀疏的大部分内容，节约了存储空间。一些图中的基本概念：

- 强连通有向图：任意两个节点 U 和 V 之间存在 U 到 V 以及 V 到 U 的路径
- 弱连通有向图：任意两个节点 U 和 V 之间存在 U 到 V 或者 V 到 U 的路径
- 强连通分量：有向图中的强连通子图

第 2 章 传统的图机器学习方法

传统的机器学习通过手工提取特征来完成图机器学习中的 node-level, link-level 和 graph-level 的一些预测, 目标是对于给定的图 $G = (V, E)$ 学习出一个从 V 映射到 \mathbb{R} 上面的函数

2.1 节点层面的任务

图机器学习在节点层面的主要任务是对节点进行分类和学习图的结构特征, 主要有:

- 节点的度数
- 节点的中心 (node centrality)
- 聚类系数
- graphlets

2.1.1 节点的 Centrality

定义 2.1

节点中心 node centrality 是在一个图中表示出不同节点的重要性, 而不是简单的出入度数。一个节点 v 如果与一些很重要的节点相邻那么这个节点也很重要, 一个节点的 centrality 可以这样计算:

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \quad (2.1)$$



问题在于上面的这个表达式其实是一种循环定义, 需要用另一种方法来表示。

2.1.1.1 特征向量中心

我们可以用向量的方式来表示上面的 centrality 定义式:

$$\lambda c = Ac \quad (2.2)$$

其中 A 是图 G 的邻接矩阵, 如果两个点 uv 相邻, 那么 $A_{uv} = 1$ 否则就是 0, 我们发现这时候由每个点的 centrality 组成的向量 C 实际上就是矩阵 A 的特征向量, 其中最大的特征向量总是为正并且唯一的, 因此可以用 A 矩阵的最大特征值作为图 G 的 centrality

2.1.1.2 中间性中心 Betweenness Centrality

该标准认为一个节点如果处于很多通往其他节点的最短路径上, 那么这个节点就比较重要, 因此定义:

$$c_v = \sum_{s \neq t \neq v} \frac{N_{svt}}{N_{sv}} \quad (2.3)$$

该标准下重要程度的计算方式就是对于图中的任意两个和 v 不同的节点 s, t , 计算其最短路径中包含 v 的比例并进行求和

2.1.1.3 邻近中心 Closeness Centrality

该标准认为一个节点如果拥有比较小的到其他所有节点的最短路径长度，那么它就比较重要，用 l 表示图中两个节点的最短路径长，那么：

$$c_v = \frac{1}{\sum_{u \neq v} l_{uv}} \quad (2.4)$$

2.1.2 聚类系数

定义 2.2

聚类系数可以衡量一个节点的邻居节点的连接情况，假设一个节点 v 有 k 个邻居，这些 k 个邻居之间有 n 条边，那么聚类系数可以写作：

$$e_v = \frac{n}{C_k^2} = \frac{2n}{k(k-1)} \quad (2.5)$$

聚类系数某种意义上也可以理解为记录了每个节点与其邻近节点构成的三角形的个数，即对于一个节点 v ，如果它的两个邻居 u 和 w 之间有一条边，那么 uvw 就构成了一个三角形，而这个三角形被统计到了聚类系数中去

2.1.3 Graphlets

Graphlets 是一种根连通的非同构子图。

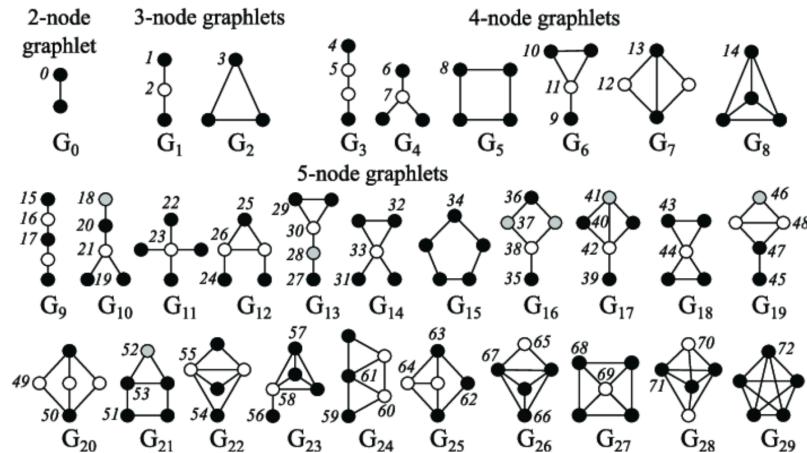
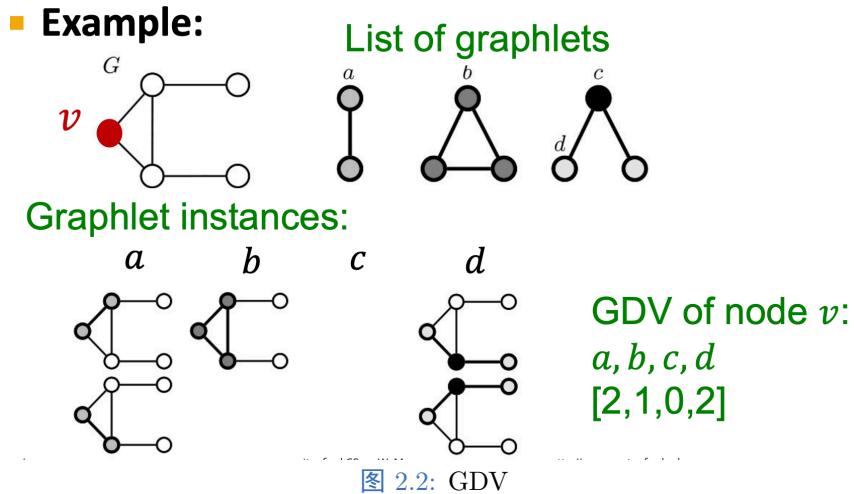


图 2.1: Graphlets

Graphlet Degree Vector(GDV): 基于 Graphlets 的节点特征，记录了每个节点所能接触到的 graphlets 的个数，和度数以及聚类系数不同，度数衡量了一个节点能接触到的边的数量，而聚类系数衡量了一个节点能接触到的“三角形”的个数，而 GDV 记录了以一个节点 v 为 root 的所有的 graphlet



GDV 提供了一个节点的局部网络拓扑结构的衡量方式

2.2 边层面的任务

图机器学习在边层面的主要任务是基于已经存在的连接来预测新的连接，因此关键的问题在于如何设计和表示一对节点的特征，一般有两种边预测的场景：

- 随机缺失了若干条边，要求将图补全
- 随时间会变化的边关系：在时序数据的场景下节点和节点之间的边关系可能会随时间发生变化，这种情况叫做 evaluation

2.2.1 基于距离的特征

用两个节点之间的最短路径距离来表示节点对的特征，但是这种方式没有考虑到邻近点的重合度

2.2.2 Local Neighborhood Overlap

用共同的邻近节点的个数作为一个节点对的特征，对于一个节点 v ，用 $N(v)$ 表示所有邻近节点构成的集合，那么有这样几种标准：

定义 2.3

共同邻居：

$$|N(u) \cap N(v)| \quad (2.6)$$

定义 2.4

Jaccard 系数：

$$\frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \quad (2.7)$$

2.2.3 Global Neighborhood Overlap

局部的 neighborhood overlap 在两个节点没有共同邻居的时候始终为 0，但这两个节点仍然有可能存在一定的关系，而 Local Neighborhood Overlap 这些关系的特征，因此就需要

Global Neighborhood Overlap

定义 2.5 (Katz index)

记录两个节点之间所有的路径数，可以使用邻接矩阵来计算，假设临界矩阵为 A ，其中如果 u 和 v 相邻那么 $A_{uv} = 1$ ，这样一来 A_{uv}^k 就可以表示是否有连接 uv 的长度为 K 的路径，因此可以用下面的共识来计算这一特征量：

$$S_{uv} = \sum_{l=1}^{\inf} \beta^l A_{uv}^l \quad (2.8)$$



而 Katz index 矩阵可以用如下方式来计算：

$$S = \sum_{i=1}^{\inf} \beta^i A^i = (I - \beta A)^{-1} - I \quad (2.9)$$

2.3 图层面的任务

目标是提取整张图整体的特征，而传统机器学习中经常使用 kernel 来进行特征的提取，图也可以使用 kernel 来进行特征的提取，其核心是一个图特征向量 $\phi(G)$

2.3.1 BoW

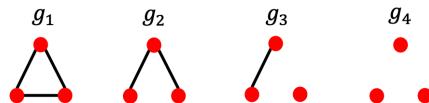
图 kernel 的关键想法是基于 Bag-of-Word(BoW)BoW 原本是指使用单词出现的次数来作为文档的特征，而将其扩展到图上就可以将节点作为“单词”，即可以计算图中各种不同度数的节点的数量，并将结果作为一个多维的向量输出，向量的每一个维度 d 的值代表了图中度数为 d 的节点总数

2.3.2 Graphlet 特征

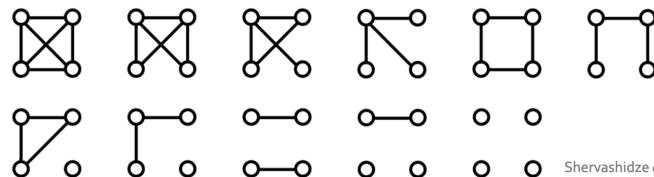
可以计算图中不同的 graphlet 的个数来表示图的特征，但是这里的 graphlet 和节点层面的不太一样，不需要完全连接，也没有根的概念。

Let $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$ be a list of graphlets of size k .

- For $k = 3$, there are 4 graphlets.



- For $k = 4$, there are 11 graphlets.



Shervashidze et al., AISTATS 2011

图 2.3: Graphlet 特征

对于一个给定的图 G ，用一个 graphlet 表来定义 graphlet 计数向量 $f_G \in \mathbb{R}^{n_k}$ ，则：

$$(f_G)_i = \#(g_i \subset G) \quad (2.10)$$

graphlet kernel 可以定义为:

$$K(G, G') = f_G^T f_{G'} \quad (2.11)$$

如果两张图的 size 不同会引起结果的失真, 那么就可以进行正则化操作, 令:

$$h_G = \frac{f_g}{\sum f_G} \rightarrow K(G, G') = h_G^T h_{G'} \quad (2.12)$$

这种 kernel 的问题在于计算 graphlet 的个数的时间复杂度是非常高的, 并且是 NP-hard 的

第3章 节点嵌入

图表示学习的目标是提供一个统一的图特征提取方式，对于不同的图机器学习任务都可以使用，而节点嵌入就是指将图中的节点映射到嵌入空间中，用一个稠密的向量来表示不同的节点，而向量的相似度又决定了节点在图中的相似度，也就是说将整个网络进行了编码。

3.1 编码器和解码器

3.1.1 关键组件

节点嵌入的目标就是对节点进行编码并映射到嵌入空间中，使得两个节点在嵌入空间中的相似度近似于节点在图中的相似度，而相似度和嵌入向量的形式都是需要定义的。因此节点嵌入的两个关键的组件就是编码器和相似度函数：

$$\text{similarity}(u, v) = z_u^T z_v \quad (3.1)$$

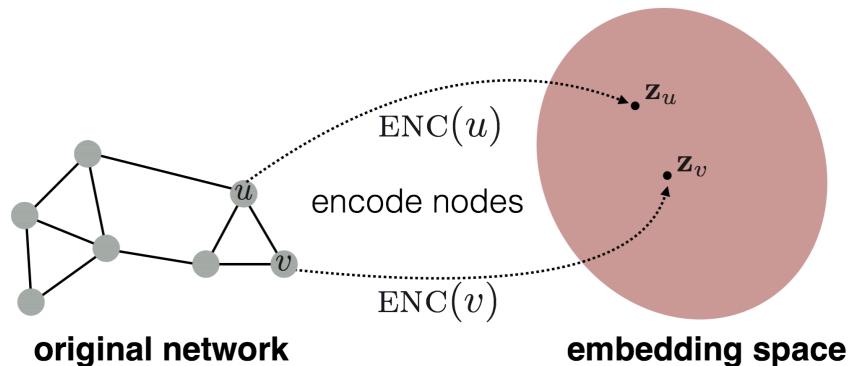


图 3.1: 编码器

3.1.2 浅编码 Shallow Encoder

最简单的编码器的形式就是一个简单的嵌入映射，即将节点通过矩阵运算直接转化成对应的嵌入向量，可以表示为：

$$\text{ENC}(v) = \mathbf{Z}v \quad (3.2)$$

- 其中 \mathbf{Z} 就是一个 $d \times |V|$ 维度的矩阵，存储了每个节点的 d 维嵌入向量，而 v 就是一个 0-1 向量，除了对应的节点那一列是 1 以外都是 0
- 我们需要学习的就是 \mathbf{Z} 矩阵，这种编码器下面每个节点都映射到一个单独的嵌入向量中

3.2 随机游走 Random Walk

3.2.1 什么是随机游走

定义 3.1

随机游走是一种用来定义图节点相似度的方法， z_u 表示图节点 u 的嵌入向量，而概率 $P(v|z_u)$ 表示在节点 u 的随机游走中遇到节点 v 的概率。

随机游走的过程即每次随机选择当前节点的一个邻居并“走”到这个邻居的位置上不断重复的过程，这个过程中将产生一个随机的节点序列，称为图的随机游走。而用随机游走定义的相似度就是 u 和 v 出现在同一个随机游走中的概率。



这种方式计算相似度需要以下几个步骤：

- 根据随机游走的结果优化嵌入函数并进行编码
- 使用一定的决策策略 R 来计算从 u 出发的随机游走经过 v 的概率

3.2.2 为什么需要随机游走

- 可表达性：随机游走是一种灵活并且随机的相似度定义，并且包含了局部信息和更高阶的图中节点关系
- 高效性：不需要在训练的过程中考虑所有的节点对，只需要考虑在随机游走中出现的节点对
- 随机游走是一种无监督的特征学习

3.2.3 随机游走的优化

随机游走的目标是让学习到的嵌入向量中，相近的向量在图中更接近，对于一个节点 u 可以定义它在某种选择策略 R 下的随机游走中发现的邻居节点构成的集合是 $N_R(u)$ ，对于一个给定的图 $G = (V, E)$ ，我们的目标是学习出一个映射函数 $f(u) = z_u$ ，根据极大似然估计，我们的目标函数可以确定为：

$$\max_f \sum_{u \in V} \log P(N_R(u)|z_u) \quad (3.3)$$

即对于给定的节点 u ，我们希望通过随机游走中的表现来学习其特征的表示，而虽有游走可以进行一系列的优化，包括：

- 进行一个较短的固定长度的随机游走
- 对于每个节点 u 的邻居节点，允许邻居节点集合中出现重复的节点，出现的越多表明相似度越高，因此最大化上述目标函数可以等价于最小化下面的表达式：

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|z_u)) \quad (3.4)$$

而概率 $P(v|z_u)$ 可以用 softmax 函数来进行参数化，选用 softmax 函数的原因是因为指数运算避免了负概率的出现，并且使得不同节点的相似度区分变得更加明显

$$P(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \quad (3.5)$$

但是用上述办法来计算目标函数的话复杂度是非常高的，，可以采用负采样的方式来近似计算损失函数，这里用到了 sigmoid 函数来近似计算：

$$\log\left(\frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}\right) \approx \log(\sigma(z_u^T z_v)) - \sum_{i=1}^k \log(\sigma(z_u^T z_{n_i})) \quad (3.6)$$

- 这种近似方法不计算全部节点而是只采样了 K 个随机的负样本，并且用 sigmoid 函数来近似指数运算，这里的 k 个 negative nodes 按照其度数成正比的概率进行选取
- 在得到了目标函数的近似形式之后，我们可以采用随机梯度下降法来对目标函数进行优化，定义

$$\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|z_u)) \quad (3.7)$$

- 对于一个节点 i，和所有的节点 j，计算其导数 $\frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$
- 更新每一个向量 j 直到收敛

$$z_j = z_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial z_j} \quad (3.8)$$

3.3 node2vec

现在的问题就变成了如何确定随机游走的策略，上面已经提到的策略有固定长度，没有偏见的选择策略，node2vec 原论文发表在在 KDD16 中，提出了一种名为 node2vec 的图节点嵌入算法，采用一种有偏见的随机游走策略，这种策略更加灵活并且达到了局部和全局的平衡。

常见的采样邻近节点的方式有 BFS 和 DFS，而 BFS 更注重局部的邻居结构，DFS 则更偏向于提取全局特征

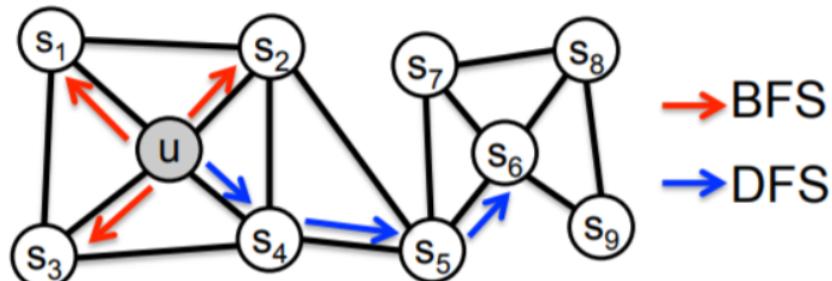


图 3.2: BFS 和 DFS

3.3.1 有偏见的随机游走

node2vec 中定义了一种二阶的随机游走策略，并且给每种不同类型的点分配了不同的权重，对于一个节点 v，假设随机游走到 v 之前的一个节点是 x，那么在 v 的所有邻居节点中，不同的节点按照到节点 x 的最短路径的边数分成了三类，即 0(节点 x 本身)，1(和 vx 两个节点都相邻) 和 2(和 v 相邻但是和 x 不相邻)，随机游走的概率可以定义为：

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx} \quad (3.9)$$

而这里的 bias 被定义为：

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \quad (3.10)$$

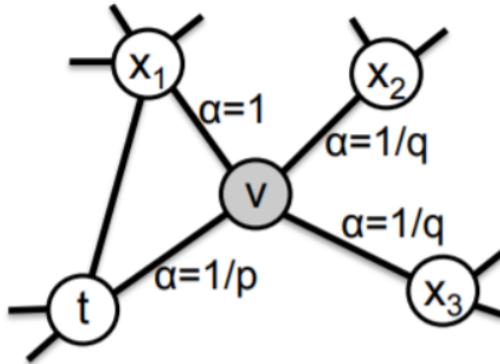


图 3.3: 带偏见的随机游走

3.3.2 node2vec 算法框架

Algorithm 1 The node2vec algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d , Walks per node r , Walk length l , Context size k , Return p , In-out q)
 $\pi = \text{PreprocessModifiedWeights}(G, p, q)$
 $G' = (V, E, \pi)$
Initialize $walks$ to Empty
for $iter = 1$ **to** r **do**
 for all nodes $u \in V$ **do**
 $walk = \text{node2vecWalk}(G', u, l)$
 Append $walk$ to $walks$
 $f = \text{StochasticGradientDescent}(k, d, walks)$
return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u , Length l)
Initialize $walk$ to $[u]$
for $walk_iter = 1$ **to** l **do**
 $curr = walk[-1]$
 $V_{curr} = \text{GetNeighbors}(curr, G')$
 $s = \text{AliasSample}(V_{curr}, \pi)$
 Append s to $walk$
return $walk$

图 3.4: node2vec 算法伪代码

关于 node2vec，我在个人博客中专门写了一篇原论文的阅读笔记，可以[点击此处查看](#)。

3.4 图嵌入

定义 3.2 (图嵌入)

图嵌入是将整张图或者子图映射到嵌入空间中，用一个向量来表示，可以用于图分类、图聚类等任务中（就是将得到的向量用作图的特征，然后使用分类、聚类等机器学习算法），常见的方法有以下三种：



- 方法 1: 将图嵌入等价于图中所有节点的嵌入向量之和

$$Z_G = \sum_{v \in G} z_v \quad (3.11)$$

- 方法 2: 在图中引入一个虚拟节点代表整个图 (子图) 并进行嵌入

- 方法 3: 匿名游走嵌入:

- 想法 1: 对匿名游走进行采样并且用每种匿名游走的发生次数来表示一个图, 也就是说用这些随机游走发生的概率来表示整张图, 选定一个长度 L 之后, 建立一个向量代表所有长度为 L 的随机游走发生的概率来代表这张图
- 想法 2: 将匿名游走的嵌入结果进行合并

第 4 章 链接分析：PageRank 算法

这一节的内容主要从矩阵的角度来进行图的分析和学习，我们可以将互联网看成一张巨大的图，里面的网页就是图中的一个个节点，而网页可以通过超链接可以跳转到别的网页，称为链接 link，可以把这种关系作为图中的有向边，因此互联网中的网页构成一张大而稀疏的有向图，可以用一个邻接矩阵来表示。类似的结构有论文引用图等等。

但是图中每个节点并不都是同等重要的，可以通过一系列链接分析的方法来分析出不同节点的重要性，一般认为一个网站如果有很多链接，那么它往往就比较重要。而出链接和入链接又是两种不同的链接，又应该有不同的考虑。PageRank 算法认为被重要的网页指向的网页也往往更重要。

4.1 PageRank 算法细节

4.1.1 节点的权重

我们可以在网络图中定义 d_i 是一个节点的出度，而节点的权重就可以表示为：

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i} \quad (4.1)$$

我们可以将这种表示形式矩阵化，用一个矩阵 M 来表示各个节点之间的权重关系，那么根据上面的定义可以得知：

$$M_{ij} = \begin{cases} \frac{1}{d_j} & \text{if } i \rightarrow j \\ 0 & \text{otherwise} \end{cases} \quad (4.2)$$

这个矩阵 M 每一列的和都是 1，我们可以用一个向量 r 来表示每个网页的重要程度，那么就有：

$$r = Mr \quad (4.3)$$

4.1.2 图中的随机游走

在任何一个时间 t 时假设访问到了网页 i 中，下一个时刻 $t+1$ 则访问 i 指向的其中一个网页 j ，这样就构成了一次随机游走，用 $p(t)$ 向量来表示 t 时刻每个网页被访问到的概率，那么这样一来就有：

$$p(t+1) = Mp(t) \quad (4.4)$$

我们发现矩阵的权重向量 r 也可以表示随机游走的分布情况，进一步我们发现 r 其实就是矩阵 M 的特征向量，因此 PageRank 实际上就是矩阵 M 最大的特征向量，我们一般用幂法可以得到矩阵 M 的最大特征向量。

4.2 求解 PageRank

可以给 pagerank 赋予一定的初始值，然后通过公式 $r = Mr$ 不断迭代直到 r 的变化小于一定的阈值之后才结束，得到最终的 pagerank 的结果，这种方法也就是幂法，求出的结果实际上是 M 的特征之中范数最大的那一个。但是问题在于：

- 有一些页面是 dead end，没有跳转到其他网页的链接，这可能会导致“泄漏”
- Spider-Trap 问题：所有的外部链接都在一个组内，即随机游走会陷入一个循环中

- 这些情况都会导致上述计算方法最后不收敛，因此要想办法解决这个问题
解决方法是，对于 dead end 问题，可以重新调整矩阵 M 中的内容：

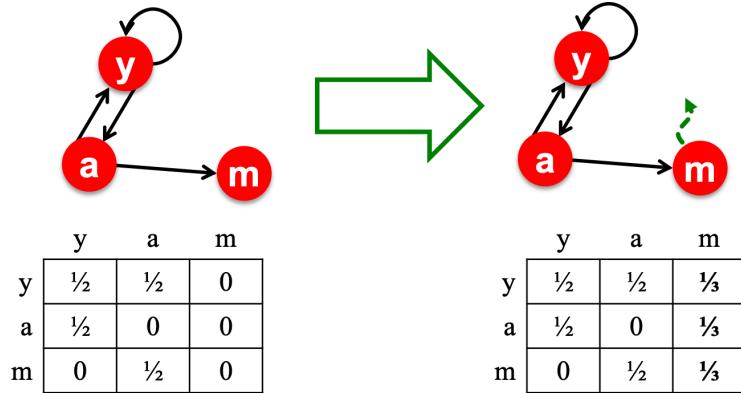


图 4.1: dead end 问题解决方法

而对于 Spider-Trap 问题，可以在每次做选择的过程中以一定的概率跳转到随机的网页中去，这样就可以从循环中跳出来

$$r_j = \beta \sum_{i \rightarrow j} \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N} \quad (4.5)$$

Google 矩阵上面的 spider-trap 问题的迭代形式：

$$G = \beta M + (1 - \beta) \left[\frac{1}{n} \right]_{n \times n} \quad (4.6)$$

4.3 个性化的 PageRank

通过几个特定的节点来衡量所有节点的 rank，可以用带 restart 的随机游走高效计算，这是考虑到了不同用户之间的浏览偏好往往不同，不能用全局的所有节点作为衡量标准，这样一来可以实现个性化的推荐

第 5 章 消息传递和节点分类

这一节内容主要关注对于一个给定的网络结构，一些节点已经做好了标注，那么我们如何对图中的剩余节点进行标注，就比如一张图中已知部分节点是可疑的，另一部分节点是可信的，那么我们应该如何判定剩下那些没有标注的节点是否可信呢？

- 这类任务被称为半监督的节点分类问题
- 之前已经介绍了节点嵌入的方法可以识别相似的节点，而这一节主要介绍的方法是消息传递

一般我们认为网络中的关系是具有相关性的，也就是说相似的节点会互相连接，并且相近的节点可能属于相同的类型，

- 同质性 Homophily：指的就是相同或者类似的节点个体会和其他的节点相互连接
- 影响：网络中的连接关系可能会影响到单个节点，即节点可能会被与之相互连接的节点给带偏

这样的半监督学习方式经常被用在文本分类，语音标记，链接预测，光学字符识别，节点异常检测。这一节的内容将主要集中精力介绍三种实现半监督图学习的技术：

- 关系分类 Relational classification
- 可迭代分类 Iterative classification
- 信任传播 Belief propagation

5.1 关系分类

关系分类的基本想法：节点 v 的类概率 Y_v 是一个节点附近节点的类别的加权平均（这里我们只考虑二分类问题，正反类别的标签分别用 1 和 0 来表示），而对于已经有标记的节点 v ，其 Y_v 可以直接设置成其真实类别，而对于没有标记的节点可以将其初始化为 $Y_v = 0.5$ ，然后对所有的节点进行类似于加权平均的更新知道收敛或者达到最大迭代次数上限。

$$P(Y_v = c) = \frac{1}{\sum_{(v,u) \in E} A_{v,u}} \sum_{(v,u) \in E} A_{v,u} P(Y_u = c) \quad (5.1)$$

其中 A 表示每条边的权重，这种方法的问题在于，收敛是没有保证的，并且模型没有用到节点的特征信息。

5.2 迭代分类

相比于关系分类，迭代分类会使用节点 v 的一系列特征构成的特征向量 f_v 和邻近节点的标签 z_v ，而这种分类方式需要训练两个分类器：

- $\phi_1(f_v)$ 通过节点的特征来预测节点的标签
- $\phi_2(f_v, z_v)$ 通过特征向量和邻近节点的标记来共同预测节点的标签

迭代分类模型的训练过程如下：

- 首先训练一个基于二维特征向量的分类器，用邻近节点（区分出和入节点）的特征向量作为 z_v
- 然后按照步骤训练出两个分类器，然后用测试集来测试分类器的效果，并对关系特征 z_v 和标签进行更新
- 进行迭代更新，直到收敛或者达到次数上限

训练两个分类器只是第一阶段，第二阶段需要进行迭代训练直到结果收敛或者训练结束，但是结果是否收敛依然没有保证。

5.3 协作分类：信任传播机制

信任传播机制是一种动态规划算法，通过一个节点和节点之间互相“通信”和传递消息的迭代过程，当节点直接达到共识的时候就可以计算出最后的结果

5.3.1 一个简单的消息传播的例子

举一个简单的例子来说明消息传递的作用，比如我们需要在一个图中计算节点总数，并且每个节点只能和与之相邻的节点之间进行通信，简单的消息传递算法的实现如下：

- 定义节点的顺序形成一条完整的路径
- 每次向下一个节点来传播当前统计到的节点个数，最后一个节点要发送出去的数据就是结果
- 这样的算法也可以在树状的结构中进行，区别就是要从叶节点将信息汇总到根节点

5.3.2 循环消息传递算法

加入一个节点 i 指向 j ，那么 i 将会给 j 传递消息，而传递什么消息需要根据 i 从指向 i 的节点中获得的信息来决定，循环消息传递算法中定义了这样几个关键的变量：

- 标签势能矩阵， $\Phi(Y_i, Y_j)$ 表示对于一个节点 j ，如果给定它的邻居节点 i 属于类别 Y_i 时，节点 j 属于类别 Y_j 的概率
- 先验信任 $\phi(Y_i)$ 表示节点 i 属于 Y_i 类别的概率
- i 向 j 传递的消息可以记成 $m_{i \rightarrow j}(Y_j)$ ，这表示 i 估计 j 的最终类型是 Y_j 在循环消息传递算法中，传递的消息可以表示为：

$$m_{i \rightarrow j}(Y_j) = \sum_{Y_i \in \mathcal{L}} \Phi(Y_i, Y_j) \phi(Y_i) \prod_{k \in N_i, k \neq j} m_{k \rightarrow i}(Y_i) \quad (5.2)$$

- 这种计算方式的意思是，如果 i 预测 j 的最终标签是 Y_j ，那么就需要考虑节点 i 所有可能的标签，并且对于每一个标签计算节点 i 收到的信息是节点 i 属于类别 Y_i 进行求和，我们可以记

$$b_i(Y_i) = \phi(Y_i) \prod_{k \in N_i, k \neq j} m_{k \rightarrow i}(Y_i) \quad (5.3)$$

如果我们遇到的图是有环的，那么这样一来节点就不存在一个确定的顺序，但我们依然可以用上述算法来计算信任的传播。当有顺序的时候可以从任意一个节点开始，然后依次更新其邻居节点的信息，但是当图有环的时候，从不同的子图中获得的信息就不是独立的了，这个时候信任传递依然可以使用，但是会出现循环传递的情况，可能会导致不收敛的情况，但是在实际操作中，这个算法依然比较好用。

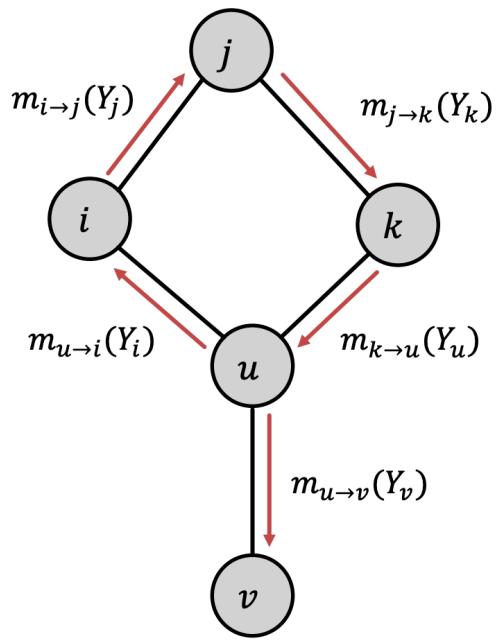


图 5.1: dead end 问题解决方法

第 6 章 图神经网络 GNN

前面几节内容主要介绍了节点嵌入的概念，也就是说我们可以将一个图中的节点映射到一个的 d 维向量上，而这种映射方式使得相似的节点对应的向量更接近，但主要的问题还是，我们如何学习出一个映射函数 f ，而图嵌入的两个核心组件是编码器和相似度函数，之前也介绍了比较 naive 的 shallow 编码器，而图神经网络 GNN 提供了一种基于深度学习的节点嵌入方法。

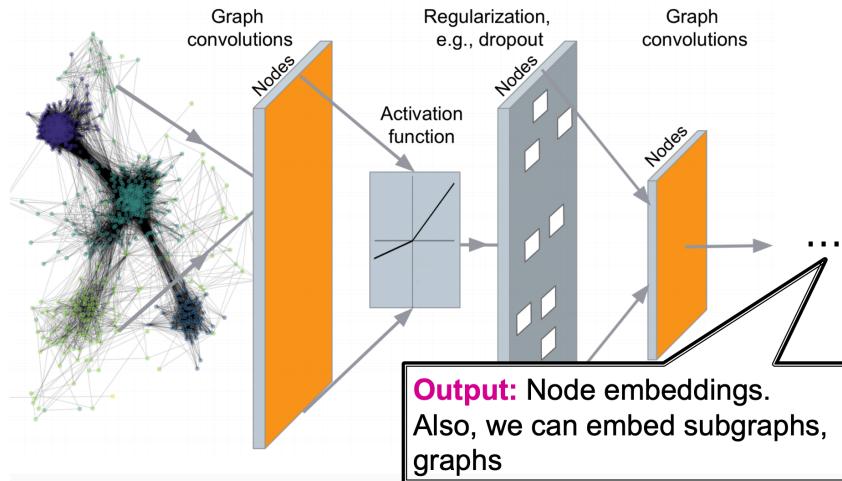


图 6.1: GNN 架构图

6.1 深度学习基础

这部分内容介绍了监督学习、损失函数、机器学习中的优化、梯度向量等基本概念，不过这些内容已经反复接触过了，因此也就不再专门整理记录。

一般的梯度下降需要求出整个损失函数的梯度，并结合设置好的学习率进行参数的更新，但是这样做的计算复杂度太高了。随机梯度下降就是每一次选择一个小批量 (minibatch) 的数据集 B 对其求梯度并进行参数的更新。

深度学习中的函数往往更为复杂，并且在反向传播进行参数更新的过程中需要用链式法则来求梯度，但总的来讲，不管是传统的监督机器学习还是深度学习，我们的优化目标都是：

$$\min \mathcal{L}(y, f(x)) \quad (6.1)$$

6.2 图深度学习

6.2.1 问题的定义与 naive approach

我们把要研究的图记为 G ，图中所有节点构成集合 V ， A 是图的接邻矩阵。用 X 表示图节点的特征矩阵，每个节点具有 m 维的特征，节点的特征可以根据图的实际情况来选取

一种很 naive 的方法是将图 G 的接邻矩阵和特征矩阵进行合并，然后放到一个深度神经网络中进行学习，但是这样一来就会有 $O(|V|)$ 数量级的参数，并且对于不同大小的图需要重新设计网络结构，这样的处理方式也使得结果对节点的顺序非常敏感

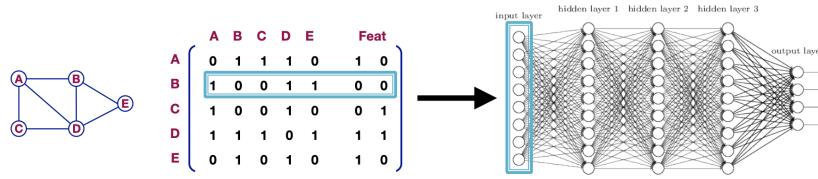


图 6.2: naive approach

6.2.2 图卷积网络的结构

联想到 CNN 通过卷积的方式来提取和融合邻近像素点的特征，图结构中也可以聚合节点的邻近节点的特征，但是相比于图像，图结构往往不具有一个固定的子结构或者滑动窗口可以用来定义图中的卷积，并且是 permutation invariant 的，我们可以使用局部的邻近节点的特征来生成图节点的嵌入向量。

- 每个节点生成了一张计算图，来表示其特征融合的过程

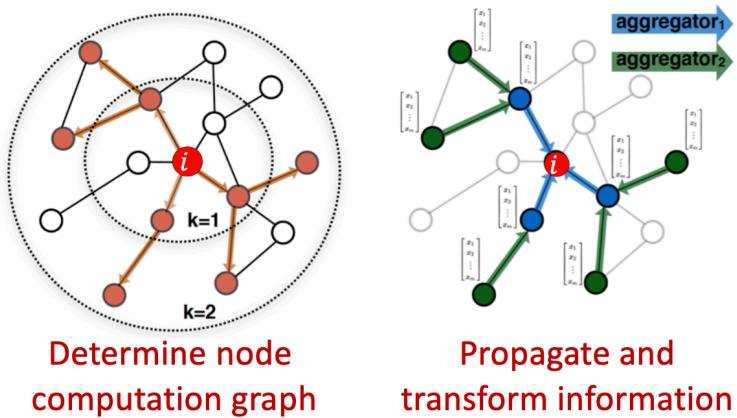


图 6.3: 计算图

- 这种神经网络可以是任意深度，每一层有节点的嵌入：
 - 第 0 层只用输入的特征表示节点的嵌入向量
 - 第 k 层使用从 k 个 way 之前得到的信息来进行节点嵌入

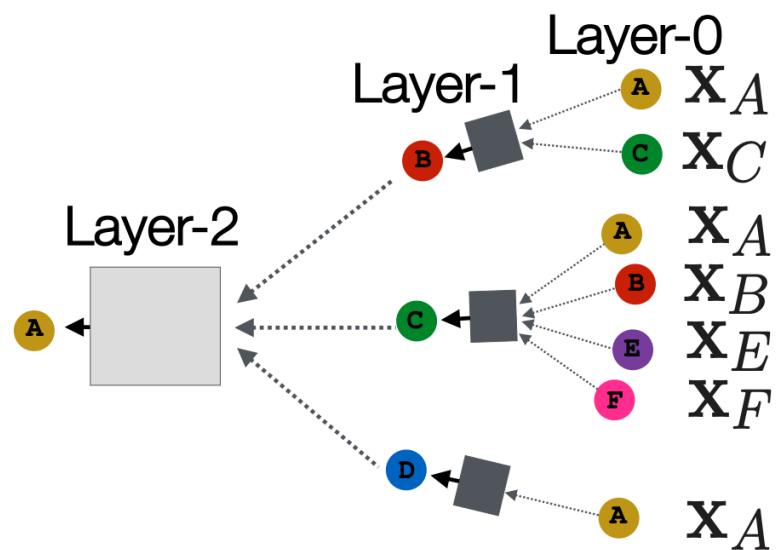


图 6.4: 信息传递的过程

上述结构只是一种最基本的框架，最主要的区别在于图中的 box 对应的内容，也就是信息聚合的方式和之后的一系列处理。常见的方法是对得到的信息求平均并放到神经网络中

$$h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \quad h_v^0 = X_v \quad (6.2)$$

- 这里的 σ 是一个非线性的激活函数
- 在得到了嵌入向量之后可以使用损失函数和随机梯度下降的方式来训练权重参数 W 和 B

我们可以将上面的表达式进行向量化来提高计算的效率，用对角矩阵 D 来表示 $D_{v,v} = |N(v)|$ ，这样一来就有 $D_{v,v}^{-1} = \frac{1}{|N(v)|}$ 并且令 $H^{(l)} = [h_1^{(l)}, \dots, h_{|V|}^{(l)}]^T$ ，这样一来上面的式子就可以表示为：

$$H^{(l+1)} = \sigma(D^{-1} A H^{(l)} W_l^T + H^{(l)} B_l^T) \quad (6.3)$$

整个过程可以分成 self translation + neighborhood aggregation 两个部分。

6.2.3 如何训练 GNN

监督学习的训练方式：定义一个 loss 函数并进行优化，相似的节点会有相近的嵌入向量，因此我们可以定义：

$$\mathcal{L} = \sum_{z_u, z_v} \text{CE}(y_{u,v}, \text{DEC}(z_u, z_v)) \quad (6.4)$$

- CE 表示交叉熵， y 是表示两个节点相似的标签，DEC 是解码器，比如可以使用内积来定义
- 相似度的定义可以用前面提到的任何方法，包括随机游走，矩阵分解等等

无监督学习的训练方式：没有节点的标签，可以使用图的结构作为监督。对于一个节点分类的问题，可以用监督学习的方式直接训练模型，使用交叉熵 loss 表示如下：

$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta)) \quad (6.5)$$

这里的 y 表示节点的标签，而 z 表示嵌入向量， θ 表示分类的权重。但是最终，我们学习到的一系列参数和模型是需要在新的图和新的节点上测试效果的，因此需要使模型拥有更好的泛化能力

6.3 经典的 GNN 模型

GNN 由一系列 GNN 层线性组合构成，而 GNN 层包含了 message 和 aggregation 等多层次的信息，当我们训练一个 GNN 的时候可以从监督学习的视角出发进行训练，也可以从非监督学习的视角出发进行训练。

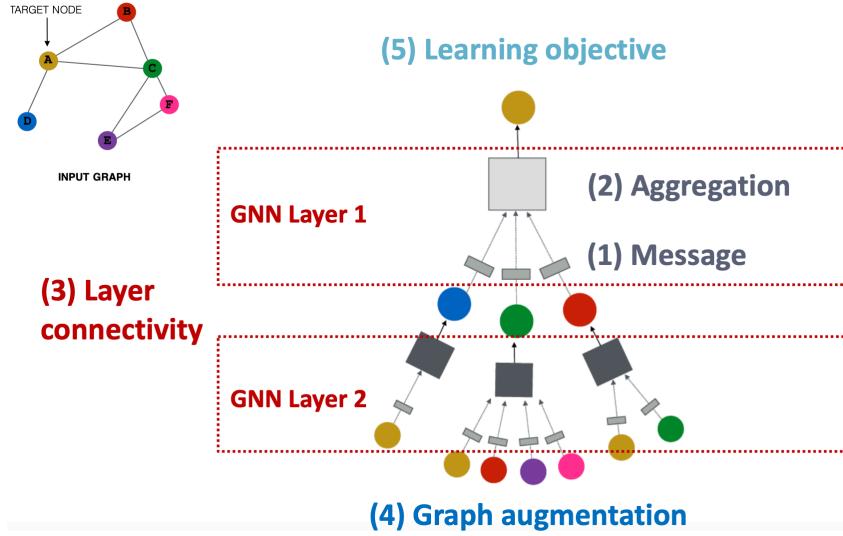


图 6.5: GNN 架构

6.3.1 GNN 的层级结构

GNN 单独的层结构将多个向量组合成一个单独的向量，需要经过两个步骤的处理，分别是 message 和 aggregation，将输入的节点嵌入向量转化成输出的节点嵌入向量。

- message 步骤：每个节点都会创建一个消息发送给附近的一系列节点，这样一来就有

$$m_u^{(l)} = \text{MSG}^{(l)}(h_u^{(l-1)}) \quad (6.6)$$

- aggregation 步骤：每个节点会收集从其他节点发来的消息并进行一定的处理，比如求和，求均值或者最大值等等。

$$h_v^{(l)} = \text{ACG}^{(l)}(m_u^{(l)}, u \in N(v)) \quad (6.7)$$

但是只有 aggregation 的话每一层的节点输出的嵌入向量都不包含其自身从上一层携带回来的特征，因此可以将 message 步骤和 aggregation 步骤得到的结果进行合并作为最终的输出结果，并且可以在两个过程中使用一些非线性的激活函数：

$$h_v^{(l)} = \text{CONCAT}(\text{ACG}^{(l)}(m_u^{(l)}, u \in N(v)), m_v^{(l)}) \quad (6.8)$$

6.3.1.1 GCN

对于图卷积网络 GCN 的层结构，上面已经介绍过了它每一层的嵌入向量更新方式：

$$h_v^{(l+1)} = \sigma \left(\sum_{u \in N(v)} W_l \frac{h_u^{(l)}}{|N(v)|} \right) \quad (6.9)$$

这个公式里面其实分成 message 和 aggregation 两个部分，其中求和就是 aggregation，而 $W_l \frac{h_u^{(l)}}{|N(v)|}$ 就是每个节点产生并广播到所有邻居节点的 message

6.3.1.2 GraphSAGE

GraphSAGE 是另一种架构的图神经网络，其核心思想是使用一个可微分的函数将邻居节点传递过来的信息应设成一个单独的 vector 每一层的更新方式是：

$$h_v^{(l)} = \sigma(W_l \times \text{CONCAT}(B_l h_V^{(l-1)}, \text{ACG}(h_u^{(l-1)}, u \in N(v)))) \quad (6.10)$$

- 这种架构的 Message 是每个邻居节点发过来的嵌入向量，通过 ACG 函数进行 aggregation，之后再和当前节点本身
- 其中聚合函数 ACG 有多种不同的选择方式，比如使用均值函数，或者进行池化（最大池化，最小池化等等），也可以使用 LSTM 来进行邻近节点的 reshuffled
- 可以在每一层都可以进行一个 L2 标准化，即 $h_v^{(l)} = \frac{h_v^{(l)}}{\|h_v^{(l)}\|_2}$ ，使用了标准化操作之后就会使得所有嵌入向量的 L2 范数统一，可以给性能带来比较大的提升。

6.3.1.3 GAT 与注意力机制

GAT 是 Graph Attention Network，即在图神经网络中加入了注意力机制，我们可以使用 α_{vu} 来表示节点 v 的邻居节点 u 的重要程度，这样的网络可以使网络在计算时将注意力集中到一些比较重要的节点上面去。

$$h_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)} \right) \quad (6.11)$$

一种 α_{vu} 的计算方式如下：

- 首先定义一个计算重要性系数的函数 a 并计算两个相邻节点的重要程度

$$e_{vu} = a(W^{(l)} h_u^{(l-1)}, W^{(l)} h_v^{(l-1)}) \quad (6.12)$$

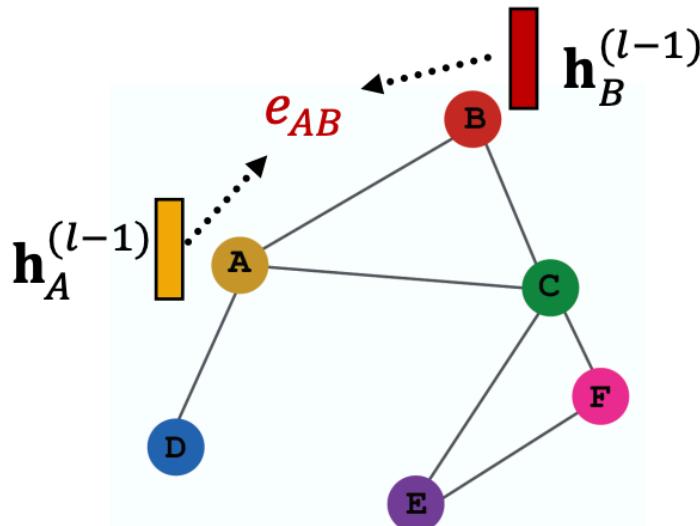


图 6.6: GAT 的注意力机制 1

- 然后使用 softmax 来计算出权重 α_{vu}

$$\alpha_{vu} = \text{softmax}(e_{vu}) = \frac{\exp(e_{vu})}{\sum_{u \in N(v)} \exp(e_{vu})} \quad (6.13)$$

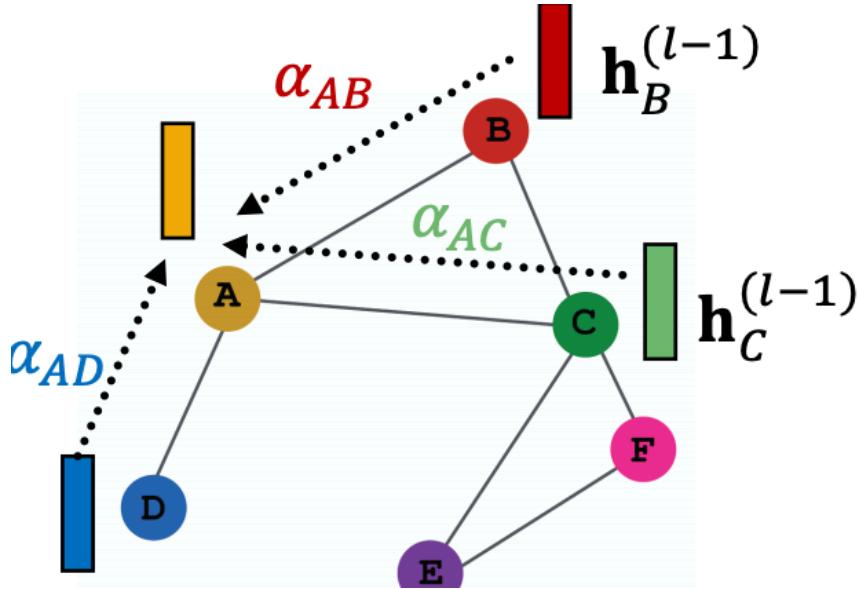


图 6.7: GAT 的注意力机制 2

- 这里的 a 函数往往是先进行嵌入向量的连接再通过一个线性函数进行运算然后输出结果

多端注意力机制：创建多个注意力分数并进行连接：

$$h_v^{(l)}[i] = \sigma \left(\sum_{u \in N(v)} \alpha_{vu}^i W^{(l)} h_u^{(l-1)} \right) \quad (6.14)$$

注意力机制最主要的优点是允许对不同的邻居节点采取不同的权重来突出一些重要的邻居，并且注意力权重的计算可以并行，提高了计算的效率，同时也提高了数据存储的效率。

6.3.2 通用的 GNN 层设计

一个通用的图神经网络的层级结构的设计如下，由线性层、批标准化层，dropout 层，激活函数层，注意力层和聚合层组成。

A suggested GNN Layer

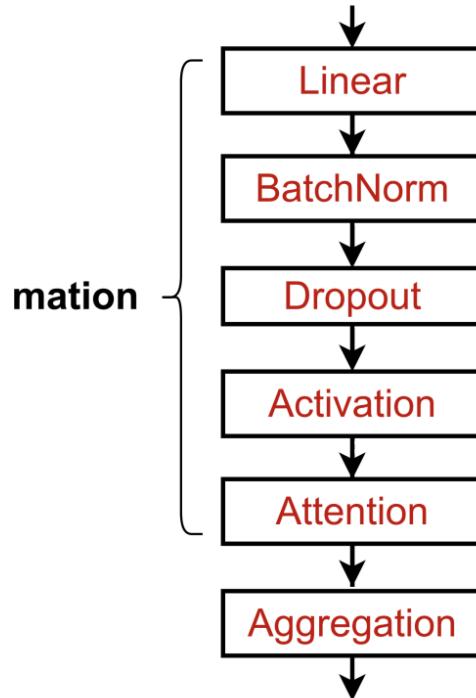


图 6.8: GNN 层的通用架构

- 批标准化: 可以是的神经网络的训练更加稳定, 做法就是求出每一层输出结果的平均值和方差, 然后进行标准化处理, 可以防止过大或者过小的数据出现而导致梯度消失或者梯度爆炸等问题
- Dropout: 用来避免神经网络过你和, 在训练的时候, 以一定概率将一些神经元设置成 0(即中途退出了计算), 在测试的时候使用所有的神经元参与到计算中, GNN 一般在线性层使用 dropout
- 激活函数: 使用一些非线性的激活函数, 将特征非线性化, 常见的非线性激活函数有 sigmoid, ReLU 等等

6.3.3 GNN 的层级架构

上面主要介绍的是 GNN 单层的设计方式和基本的 idea 以及一些常用的神经网络 tricks, 但是在设计好了若干层神经网络之后, 还需要用一定的规则将其组合起来, 在 GNN 中往往就是将多个层级进行线性的组合, 有的时候也会用的 skip connection 机制。

6.3.3.1 过平滑问题和接收域

定义 6.1 (过平滑问题)

过平滑 (Over-Smoothing) 指所有的节点嵌入向量最终收敛到了同一个值上, 这是一个非常糟糕的现象, 因为我们希望不同的节点的嵌入向量是不同的, 否则无法区别出不同的节点。



可以用接收域来解释过平滑问题出现的原因。

定义 6.2 (接受域)

接受域 (Receptive Field) 是决定一个节点的嵌入向量的节点集合，在一个 K 层的 GNN 中，每个节点的接受域包含了相距 K-hop 的邻居



随着 GNN 的层数增加，每个节点的接受域也在不断扩大，这就会导致两个节点的接受域的重合度越来越高，而接受域可以决定一个节点的嵌入向量，因此随着层数增加嵌入向量也会越来越相似，最终就会导致过平滑问题的出现。

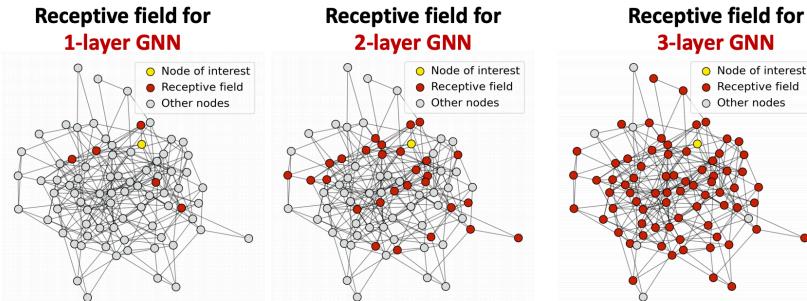


图 6.9: 接收域

6.3.3.2 解决过平滑问题

- 在添加 GNN 层的时候慎重考虑，不像 CNN 之类的网络结构，GNN 有的时候添加太多层可能不会起作用
- 因此需要先分析解决问题必要的接受域的大小，然后再将 GNN 层数设定为一个稍大于所需接受域的值，不能设置的太大，否则就会出现上述过平滑的问题
- 可以减少 GNN 的层数 (即使用 shallow GNN)，但是在每一层的 box 中增加多个线性层，也可以增加一些不传递消息的层，比如预处理层和后处理层
- 使用 skip-connect 的 trick

6.3.3.3 Skip Connection

有的时候位于更底层的 GNN 层产生的嵌入向量更能区分不同的节点，我们可以方法底层在最后的嵌入向量中的作用效果，方法就是在 GNN 中添加一些 shortcuts，也叫做 skip connection

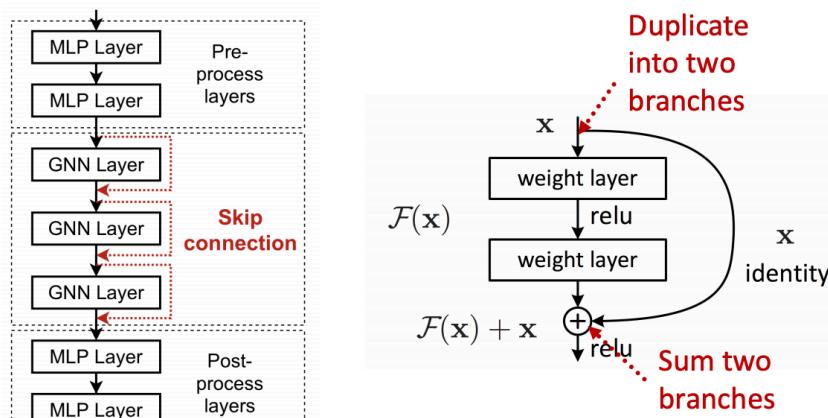


图 6.10: GNN 的 Skip Connection

为什么 skip connection 可以 work? 因为 skip 机制创建了一个混合模型, 通过 skip, 比较底层的一些计算结果直接跳过了中间一些层的计算直接作用到了更高的层次中, 这样一来就使得深的 GNN 和浅的 GNN 的计算结果进行了混合。有 skip connection 机制的 GCN 的更新方式是:

$$h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}) \rightarrow h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + h_v^{(l)}) \quad (6.15)$$

6.3.4 GNN 对图的预处理

GNN 训练之前经常需要对图进行一定的操作 (Manipulation), 因为输入的图不一定就是最优的计算图, 因为输入的图可能缺少特征, 太稀疏或者太稠密, 也有可能因为规模太大而无法在 GPU 上进行计算, 因此可以对输入的图进行一定的处理, 比如:

- 图中的特征太少的时候可以增加特征, 比如给图中的节点增加唯一 ID 标识, 使用 one-hot 向量来表示特征
- 图太稀疏可以增加虚拟的节点或者边
- 图太稠密的时候可以在消息传递的过程中进行邻居采样
- 图的规模太大的时候可以分割成若干个子图分别进行嵌入

	Constant node feature	One-hot node feature
	<p>INPUT GRAPH</p>	<p>INPUT GRAPH</p>
Expressive power	Medium. All the nodes are identical, but GNN can still learn from the graph structure	High. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	High. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	Low. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	Low. Only 1 dimensional feature	High. $O(V)$ dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

图 6.11: 图操作

第 7 章 GNN 的设计、训练和应用

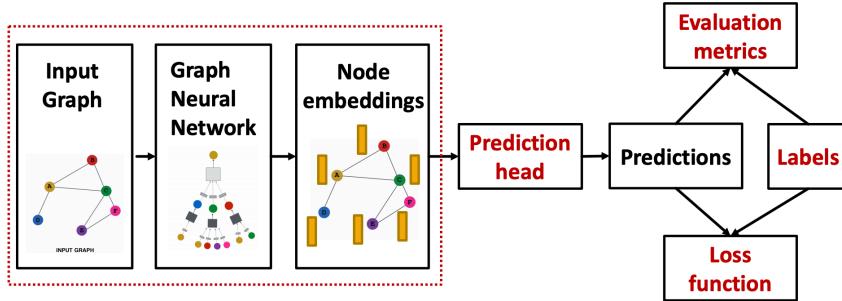


图 7.1: GNN 的 Pipeline

7.1 GNN 面对的任务

prediction head 有多种类型，包括节点级别的，边级别的和图级别的任务

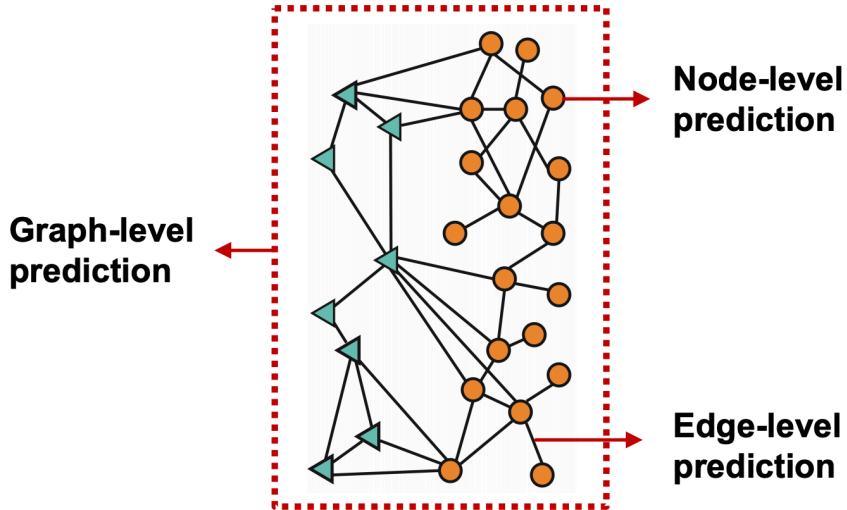


图 7.2: GNN 的任务

7.1.1 节点级别的任务

节点级别的任务可以直接使用节点嵌入来完成，假设我们通过 GNN 得到了 d 维的节点嵌入向量 $h_v^{(l)}$ ，并且需要做 k 路的 prediction(在分类问题中是 k 类的多分类任务，在回归问题中是 k 个目标的回归)，这个过程可以表示为：

$$\hat{y}_v = \text{Head}_{\text{node}} \left(h_v^{(L)} \right) = \mathbf{W}^{(H)} h_v^{(L)} \quad (7.1)$$

- 这里的 \mathbf{W} 是一个 $k \times d$ 的矩阵而 h_v 是一个 d 维的向量

7.1.2 边级别的任务

边级别的任务可以使用 2 个节点的嵌入来进行 k -way prediction，这个过程可以表示为 $\hat{y}_{uv} = \text{Head}_{\text{edge}} \left(h_u^{(L)}, h_v^{(L)} \right)$ ，而 head 的选取有如下多种方式：

- Concatenation + Linear，将两个节点嵌入进行拼接再进行线性变换

$$\hat{y}_{uv} = \text{Linear} \left(\text{Concat} \left(h_u^{(L)}, h_v^{(L)} \right) \right) \quad (7.2)$$

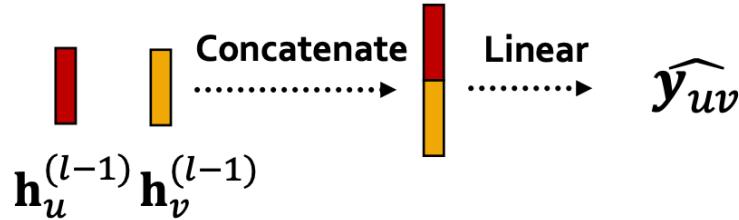


图 7.3: 嵌入向量拼接 + 线性变换

- 点积运算, 即 $\hat{\mathbf{y}}_{uv} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$ 不过只能用于 1-way 的 prediction, 可以使用 multi-head 的注意力机制来实现 k-way prediction

7.1.3 图级别的任务

图级别的任务可以使用所有节点的嵌入向量来完成, $\hat{\mathbf{y}}_G = \text{Head}_{\text{graph}} \left(\left\{ \mathbf{h}_v^L \in \mathbb{R}^d, \forall v \in G \right\} \right)$ 这里的 head 和 GNN 中的聚合函数一样, 可以使用 max, mean 和 sum 等函数将节点嵌入转化成图嵌入。这种操作也叫做图池化 (Graph Pooling), 但是这种方法很容易丢失图中的信息。解决这个问题的方法是使用层级化的池化 (Hierarchically pool), 也就是使用激活函数 + 聚合函数作为 head, 将节点划分成若干组进行池化之后再将所得结果进行池化, 这一过程可以表示为:

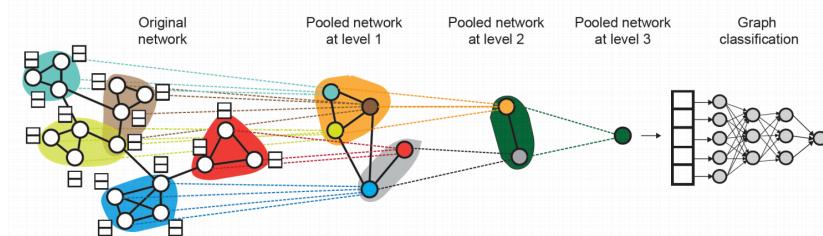


图 7.4: 图池化

7.2 监督学习和无监督学习

图学习任务中, 有监督的任务是有 label 的, 这些 label 来自于外部, 而无监督的学习任务只能使用节点自身的一些信号, 很多时候二者的差别是很模糊的。

比如我们如果使用 GNN 来预测节点的聚类系数, 看起来是一个无监督的任务, 实际上监督信息已经蕴含了图结构中 (因为对于一个确定的图而言, 其聚类系数已经可以确定, 虽然没有直接计算这些聚类系数作为监督标签, 但是聚类系数所带来的局部特性仍然表现在图结构中, 很难说 GNN 有没有学到这些隐含的监督信息), 因此很多时候将无监督学习用“自监督学习” (self-supervised) 来代替。

监督学习中的 label 可以分为节点的 label, 边的 label 和图的 label, 因此最好将要解决的问题规约到这三类 label 的监督学习中, 因为这三类任务最容易把握。无监督学习中没有标签, 但是我们可以使用图中隐含的各类信息来帮助我们完成任务。

7.3 损失函数和评价标准的选取

7.3.1 损失函数

对于分类和回归任务，需要视情况选择不同的损失函数和评价标准，分类任务的 label 是离散的数值，而回归得到的是连续的数值，因此分类任务中常常使用交叉熵作为 loss，即：

$$\text{Loss} = \sum_{i=1}^N \text{CE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = \sum_{i=1}^N \sum_{j=1}^k y_j^{(i)} \log \hat{y}_j^{(i)} \quad (7.3)$$

这里的 y_i, \hat{y}_i 分别代表真实的标签和预测得到的结果，真实的标签是一个 one-hot 的向量，而预测结果是一个 softmax 后的概率分布。而对于回归问题一般采用最小平方损失 (MSE) 作为损失函数，即：

$$\text{Loss} = \sum_{i=1}^N \text{MSE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = \sum_{i=1}^N \sum_{j=1}^k (y_j^{(i)} - \hat{y}_j^{(i)})^2 \quad (7.4)$$

7.3.2 评估标准

GNN 对于回归任务的评估标准往往采用 RMSE 和 MAE，而对于分类任务，如果是分类任务可以使用准确率，对于二分类问题还可以使用查准率和召回率以及 F1-Score

7.4 数据集的划分

训练 GNN 的过程中，需要对数据集进行一定的划分，将数据集分成训练集，测试集和验证集在图像的任务中，而图数据集的划分是比较特殊的。在 CNN 处理图像的任务中，一张图像就是一个数据点，并且图像和图像之间是互相独立的，而在图任务中，以节点分类任务为例，每个节点是一个数据点，但是数据点之间不是完全独立的，因此图数据集在划分的过程中有一定的讲究。

第 8 章 图神经网络的表示能力

我们已经了解了一系列经典的图神经网络的架构，比如 GCN, GraphSAGE, GAT 等等，这些图神经网络可以生成一系列的节点嵌入，但是我们应该怎么评估节点嵌入的表示效果（换句话说就是图神经网络的表示效果）呢？

8.1 局部结构和计算图

图中存在一定的局部结构，比如下面的这张图中，节点 1 和 2 在局部结构中是对称的，而 1 和 5 就不是，因为 1 和 2 相互交换之后图的结构不会改变。

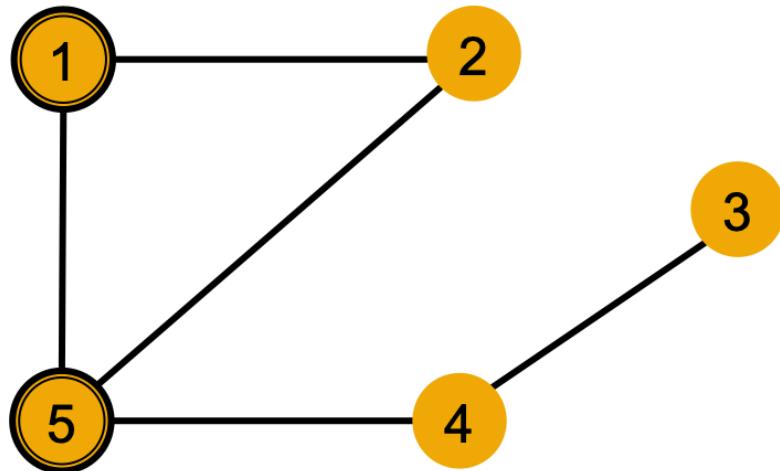


图 8.1: 计算图

而局部结构可以很好的反映出 GNN 的表达能力，一个好的 GNN 应该要能表示出节点的局部结构，可以区分对称的节点和不对称的节点，进一步，我们需要理解 GNN 是如何捕捉局部的结构信息的。

计算图可以表示出 GNN 中的每个节点如何一步步聚合其他节点的信息形成自己的嵌入向量的过程，比如上图中 1 和 2 的计算图如下：

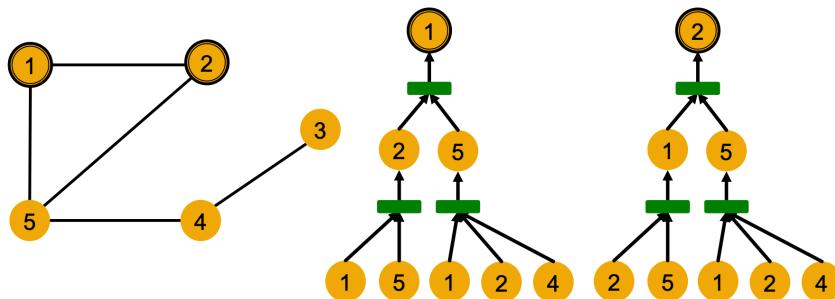


图 8.2: 计算图与节点的相似性

我们发现 1 和 2 的计算图的结构非常相似，因此最终得到的 1 和 2 的嵌入向量也是相同的，换句话说 GNN 无法区别节点 1 和 2，而局部结构类似的节点往往会有相似的计算图。

因此 GNN 的表示能力其实来自于对计算图中的一个子树结构的信息提取，一个好的 GNN 应该把不同的子树映射成不同的向量。如果一个 GNN 在每一步中都可以完全获取邻居节点的信息，那么生成的节点嵌入就可以区分不同的子树，也就是说一个表达能力强的 GNN 的聚合函数应该是一个 injective function(单射函数)

8.2 GNN 的设计

8.2.1 经典架构

本节内容的目标是归纳总结出设计高效 GNN 的一些方法论。经典的 GNN 架构中，GCN 采用了逐点的均值池化作为聚合函数，而 GraphSAGE 采用了逐点的最大池化，但是 GCN 无法区分一些数量不同但节点组成类型相同的局部结构，因为这样的不同结构在均值池化下得到的结果是相同的，与之类似的，GraphSAGE 在碰到 label 集相同的子结构的时候也无法区分，因为这个时候最大池化的结果是相同的。

8.2.2 MLP 和 GIN

一个 GNN 的聚合函数最好是一个单射函数，这种函数可以形式化的表示成非线性激活函数和线性函数的组合 $\Phi(\sum_{x \in S} f(x))$ ，我们知道一个具有足够大维数的隐层和激活函数的多层感知机就可以逼近任何连续空间中的函数，因此 GIN 架构提出了使用 MLP 作为聚合函数，也就是：

$$\text{MLP}_\Phi(\sum_{x \in S} \text{MLP}_f(x)) \quad (8.1)$$

这种 GNN 被认为是表达能力最强的消息传递类 GNN，因为采用的 MLP 理论上可以拟合任何函数，这种 GIN 实际上是 WL 核函数的神经网络形式，WL 核函数的形式如下：

$$c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right) \quad (8.2)$$

而 GIN 就是使用神经网络作为其中的 Hash 函数，因此 GIN 模型可以表示为：

$$\text{MLP}_\Phi\left((1 + \epsilon) \cdot \text{MLP}_f\left(c^{(k)}(v)\right) + \sum_{u \in N(v)} \text{MLP}_f\left(c^{(k)}(u)\right)\right) \quad (8.3)$$

第 9 章 知识图谱

到目前为止，图机器学习这门课程研究的对象的都还是只有一种类型边的图，而对于边的类型有很多种类的图（也叫做异构图，Heterogeneous Graph，边代表了关系，多种边就代表了多种更复杂的关系），常见的异构图有：

- 关系型 GCNs(RGCN)
- 知识图谱，用于知识图谱补全的嵌入

一个异构图可以定义为 $G = (V, E, R, T)$ ，其中 V 和 E 分别代表节点和边的集合，T 和 R 分别代表节点和关系的类型。

9.1 关系型图卷积网络 RGCN

图卷积网络 GCN 可以在进行一定的扩展之后可以用来处理异构图，这样的网络叫做关系型图卷积网络 RGCN，可以先回忆一下对于一个单层的 GNN，有 message 和 aggregation 两个步骤来进行节点的表示，而当图中存在多种关系，变成异构图之后，RGCN 仍然可以进行图节点嵌入的学习，处理的方式是对不同的关系类型使用不同的权重矩阵 W_{r_k} ，分成多种关系来收集 message，GCN 中的更新公式是：

$$h_v^{(l+1)} = \sigma \left(\sum_{r \in R} \sum_{u \in N(v)} \frac{1}{C_{v,r}} W_r^{(l)} h_u^{(l)} + W_0^{(l)} h_v^{(l)} \right) \quad (9.1)$$

上面的这个表达式也可以拆分成 message 和 aggregation 两个部分，还是比较容易看出来的，每个关系都有 L 个对应的权重矩阵 W，因此参数数量随着关系的变多增长的很快，经常会出现过拟合的现象，可以采取对权重矩阵 W 进行正则化的方式来避免过拟合，主要的对策方法有：

- 方法 1：使用块对角矩阵 block diagonal matrices 来降低参数的维数
- 方法 2：基学习，在不同的关系之间共享权重，将 W 矩阵表示成一系列基础的 transformations 的组合，这样一来所有的关系

可以完成一系列实体识别，连接预测等任务。

9.2 知识图谱

知识图谱是用图的形式来保存一些知识，用节点来表示实体，并且用实体的种类作为标签，用边来表示实体之间的关系。现实生活中有不少知识图谱的具体案例，知识图谱也是异构图的一种。

知识图谱中比较重要的一个任务是做图谱的补全，也就是找到知识图谱中缺失的连接关系。我们要注意到知识图谱中的关系也是有很多种类的，主要有对称的关系和可逆关系。

知识图谱的关键 idea 在于知识图谱的表示，知识图谱中的一条边可以表示为一个三元组 (h, r, t) ，分别表示头节点，关系和尾节点，实体和关系可以建模成一个 d 维空间中的嵌入向量，对于一个给定的三元组，我们的目标是 (h, r) 的嵌入向量必须要和 t 的嵌入向量尽可能接近，因此问题也就变成了如何对 (h, r) 进行嵌入与如何定义“接近”。这个过程也被叫做知识图谱嵌入，而知识图谱嵌入，存在很多经典的学习算法。

9.2.1 知识图谱中的关系模式

知识图谱中有一系列各式各样的关系，比如：

- 对称关系: $r(h, t) \rightarrow r(t, h)$, 同样的还有反对称关系
- 可逆关系: $r_1(h, t) \rightarrow r_2(t, h)$
- 组合关系 (可传递关系): $r_1(x, y) \cap r_2(y, z) \rightarrow r_3(x, z)$
- 一对多关系, 一个实体对应多个实体

9.2.2 TransE

一种最简单的评估接近程度的方式，评估函数是：

$$f_r(h, t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{L1/L2} \quad (9.2)$$

TransE 的算法伪代码为：

```
Algorithm 1 Learning TransE
input Training set  $S = \{(h, \ell, t)\}$ , entities and rel. sets  $E$  and  $L$ , margin  $\gamma$ , embeddings dim.  $k$ .
1: initialize  $\ell \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$  for each  $\ell \in L$  Entities and relations are initialized uniformly, and normalized
2:  $\ell \leftarrow \ell / \|\ell\|$  for each  $\ell \in L$ 
3:  $e \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$  for each entity  $e \in E$ 
4: loop
5:  $e \leftarrow e / \|e\|$  for each entity  $e \in E$ 
6:  $S_{batch} \leftarrow \text{sample}(S, b)$  // sample a minibatch of size  $b$ 
7:  $T_{batch} \leftarrow \emptyset$  // initialize the set of pairs of triplets Negative sampling with triplet that does not appear in the KG
8: for  $(h, \ell, t) \in S_{batch}$  do
9:    $(h', \ell, t') \leftarrow \text{sample}(S'_{(h, \ell, t)})$  // sample a corrupted triplet  $d$  represents distance (negative of score)
10:   $T_{batch} \leftarrow T_{batch} \cup \{(h, \ell, t), (h', \ell, t')\}$ 
11: end for
12: Update embeddings w.r.t.  $\sum_{((h, \ell, t), (h', \ell, t')) \in T_{batch}} \nabla [\gamma + d(\mathbf{h} + \ell, \mathbf{t}) - d(\mathbf{h}' + \ell, \mathbf{t}')]_+$  Contrastive loss: favors lower distance (or higher score) for valid triplets, high distance (or lower score) for corrupted ones
13: end loop
```

图 9.1: TransE 算法

9.2.3 TransR

TransE 模型可以将任意的关系转换到相同的嵌入空间，而 TransR 可以将节点转换成 d 维的嵌入向量，必将每个关系转换成 k 维的关系空间向量，并给出一个从实体空间到关系空间的投影空间 M

- 同样地也可以对对称关系，反对称关系，1 对多关系，可逆关系进行建模
- 但是 TransR 不能对组合关系 (可传递关系) 进行建模

9.2.4 Bilinear Modeling

TransE 和 TransR 的知识图谱建模方式都是用距离作为 scoring function 的，而 Bilinear 建模中，将实体和关系都表示成 k 维的向量，称为 DistMult，并采用向量乘积来作为 scoring function，事实上这个 score function 类似于 h 和 t 的 cosine 相似度：

$$h_r(h, t) = \langle h, r, t \rangle = \sum_{i=1}^k h_i r_i t_i \quad (9.3)$$

这种建模方式支持：

- 一对多模型的建模：向量的投影相等即可
- 对称关系：向量的内积可以换
- 问题是不能表示反对称关系、逆关系和组合关系

9.2.5 DisMult

一种使用语义匹配度作为打分函数的 KGE 方法，其打分函数为：

$$f_r(h, t) = \langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle = \sum_i \mathbf{h}_i \cdot \mathbf{r}_i \cdot \mathbf{t}_i \quad (9.4)$$

这种打分函数可以理解为 cosine 相似度，并且可以很好地刻画一对多的关系，但是不能刻画可逆的关系。

9.2.6 ComplEx

基于 DisMult 方法，将嵌入向量表示在复数空间中，其打分函数为：

$$f_r(h, t) = \operatorname{Re} \left(\sum_i \mathbf{h}_i \cdot \mathbf{r}_i \cdot \bar{\mathbf{t}}_i \right) \quad (9.5)$$

这种方法可以很好地描述对称关系，反对称关系，可逆关系和一对多关系。

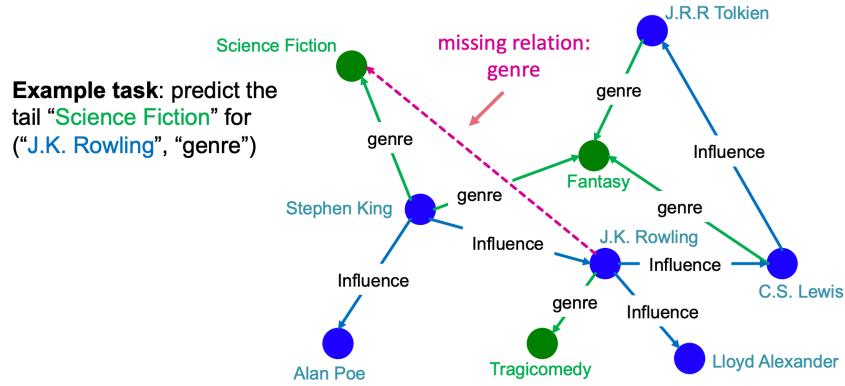
9.2.7 知识图谱嵌入算法总结

Model	Score	Embedding	Sym.	Antisym.	Inv.	Compos.	1-to-N
TransE	$-\ \mathbf{h} + \mathbf{r} - \mathbf{t}\ $	$\mathbf{h}, \mathbf{t}, \mathbf{r} \in \mathbb{R}^k$	✗	✓	✓	✓	✗
TransR	$-\ \mathbf{W}_r \mathbf{h} + \mathbf{r} - \mathbf{W}_r \mathbf{t}\ $	$\mathbf{h}, \mathbf{t}, \mathbf{r} \in \mathbb{R}^k, \mathbf{W}_r \in \mathbb{R}^k$	✓	✓	✓	✗	✓
DistMult	$\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle$	$\mathbf{h}, \mathbf{t}, \mathbf{r} \in \mathbb{R}^k$	✓	✗	✗	✗	✓
ComplEx	$\operatorname{Re}(\langle \mathbf{h}, \mathbf{r}, \bar{\mathbf{t}} \rangle)$	$\mathbf{h}, \mathbf{t}, \mathbf{r} \in \mathbb{C}^k$	✓	✓	✓	✗	✓

图 9.2: 知识图谱嵌入算法总结

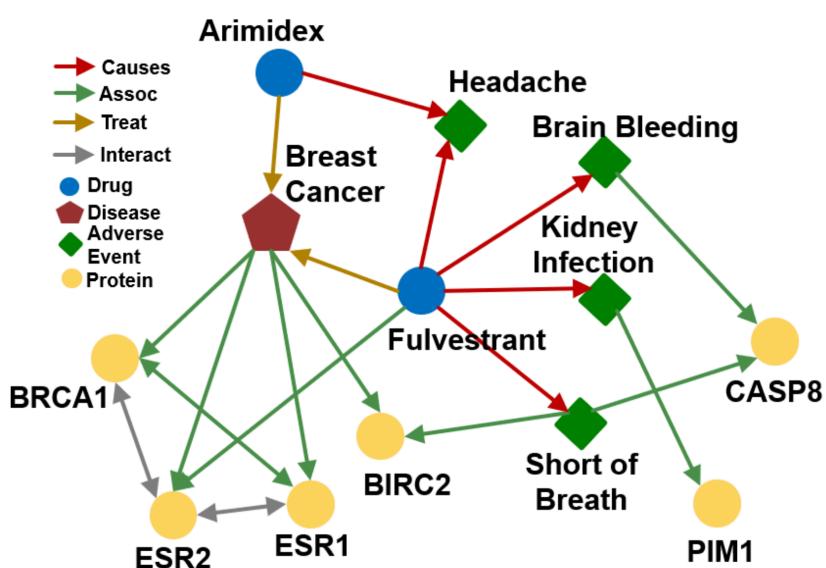
第 10 章 知识图谱推理

知识图谱中一个很常见的任务是知识图谱补全，也就是给定一个 head 和 relation，我们预测出对应的 tail，这是知识图谱推理中的推理任务 (Reasoning)



10.1 Reasoning 和 query

知识图谱推理的任务就是在知识图谱上进行 multi-hop 的推理，也就是在图上的多条路径中进行连续的推理，比如下面的生物医学知识图谱中通过症状进行疾病的推理：



而知识图谱具体的推理任务需要根据一系列的 query 来表达，也就是说在给定的不完整并且大规模的知识图谱中对输入的自然语言形式的 query 进行推理并返回结果，而 query 可以分为 One-hop Queries, Path Queries 和 Conjunctive Queries 三种，分别是单次跳转的推理，多次跳转的推理和联合推理，可以用下面的图来表示三种 query 之间的关系：

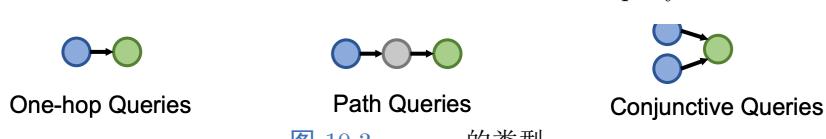


图 10.3: query 的类型

10.2 query 的形式化表述

10.2.1 one-hop query

one-hop query 实际上就是一个 head 和一个 relation 组成的，可以写成 $(h, (r))$ ，然后判断一个 tail 是否为答案，转化成知识图谱补全的问题就是判断一个三元组 (h, r, t) 是否存在于知识图谱中。

10.2.2 path query

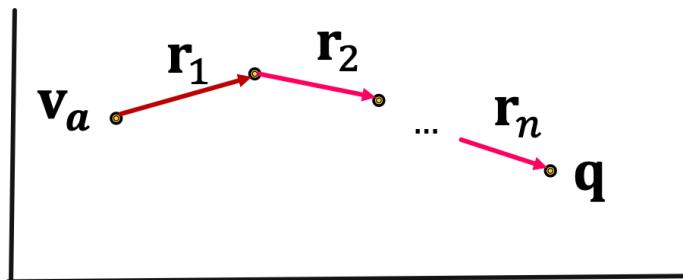
path query 实际上就是一系列的 one-hop query，可以写成 $q = (v_a, (r_1, r_2, \dots, r_n))$ ，然后需要在知识图谱中进行遍历来得到最终的结果，比如输入的 path query 是 What proteins are associated with adverse events caused by Fulvestrant，那么经过分析就可以得到 v_a 就是 Fulvestrant，而 caused, associated 就是一系列关系，然后就可以开始在知识图谱中进行一步步的查询。

虽然看起来很容易，然而知识图谱往往是不完整的，可能有很多实体和关系是缺失的，因此在推理的过程中我们很可能不能获得所有可能的结果实体。但是我们又不能把知识图谱补全之后再进行推理，因为完整的知识图谱是一个非常稠密的图，大部分三元组都会有一定概率存在，而知识图谱推理的复杂度是指数级别的，因此完整的知识图谱上的 query 的时间复杂度非常高。

因此我们希望能在不完整的知识图谱上完成基于路径的查询，同时我们希望可以隐式地对知识图谱做出一定的解释，这实际上是一种泛化后的链接预测任务。

10.3 知识图谱推理和问答

接下来就需要研究如何在知识图谱中进行 query 的问答，上一章的内容中已经提到知识图谱中的实体和关系都可以表示在向量空间中，也就是知识图谱嵌入，因此我们可以考虑将知识图谱上的问答转化到向量空间中进行，而具体的方法就是将 query 也转换成一个向量，并使用嵌入模型的打分函数来评估结果。一个查询 $q = (v_a, (r_1, r_2, \dots, r_n))$ 在向量空间中可以表示为：



$$\mathbf{q} = \mathbf{v}_a + \mathbf{r}_1 + \cdots + \mathbf{r}_n$$

图 10.4: 用知识图谱嵌入表示的推理过程

这样一来上一章提到的各种知识图谱嵌入方法，比如 TransE, TransR, DisMult 和 ComplEx 等等就可以使用到知识图谱的问答中，而上述方法中，只有 TransE 可以刻画传递关系 (composition relations) 其他几个方法都不行，因此最终用于知识图谱推理问答的只

有 TransE，而如果是更复杂的 conjunctive query，比如“What are drugs that cause Short of Breath and treat diseases associated with protein ESR2?”，它可以表示为 ((e:ESR2, (r:Assoc, r:TreatedBy)), (e:Short of Breath, (r:CausedBy))) 而查询的过程可以表示为：

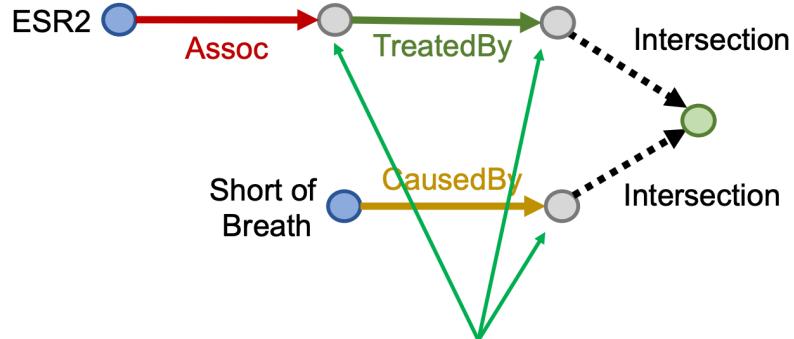


图 10.5: query 的分解

10.4 Query2Box

这一节的内容将着重介绍一种新提出的知识图谱推理方式：使用 Box Embedding 进行推理。这种方法中，一系列的 query 被表示成了一系列超矩形 (hyper-rectangles)，也叫做 box，采用 center 和 offset 两个量来描述一个 query，依然用上面那张生理医学知识图谱为例，我们可以将 Fulvestrant(一种药) 的所有不良反应都嵌入到一片矩形的区域内：

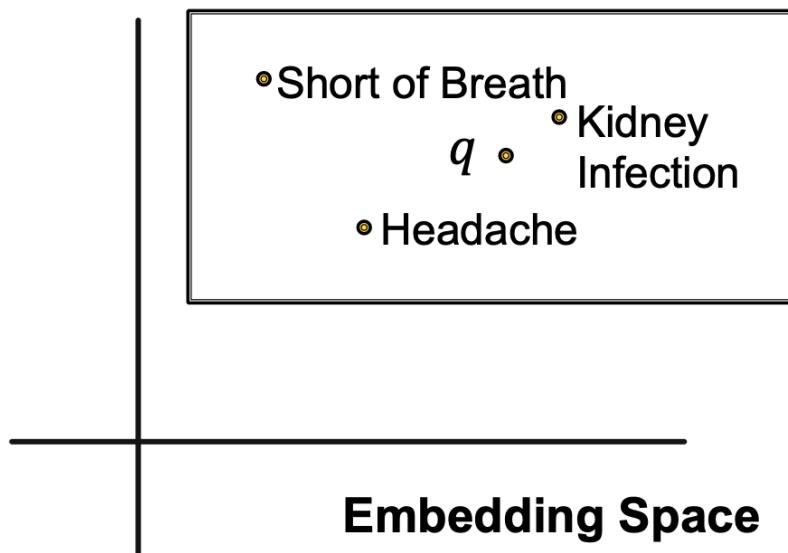


图 10.6: Query2Box

data:image/s3,anthropic-data-us-east-2/u/marker_images/0001/0100/1001/10001110/sfishman-chandramapper-0319211047/c89cfa5f1498e9753fa780a25e3ece1b.jpg</antml:image>

图 10.7: Query2Box 的投影操作

10.4.2 取交集 (Intersection) 操作

求交集是 Query2Box 的另一种基本操作, 这种操作可以求出输入的一系列 box 的交集, 也是一个 box, 根据基本常识, 这个新的 box 的中心应该接近于输入的 box 的中心, 并且 offset 不能超过输入 box 中的最小值。新的交集 box 计算方式如下:

$$Cen(q_{\text{inter}}) = \sum w_i \odot Cen(q_i) \quad (10.1)$$

这里的 w_i 是权重参数, 可以通过如下方式计算得到, 本质上是一种注意力机制:

$$w_i = \frac{\exp(f_{\text{cen}}(Cen(q_i)))}{\sum_j \exp(f_{\text{cen}}(Cen(q_j)))} \quad (10.2)$$

而 offset 可以用如下方式计算:

$$Off(q_{\text{inter}}) = \min(Off(q_1), \dots, Off(q_n)) \odot \sigma(f_{\text{off}}(Off(q_1), \dots, Off(q_n))) \quad (10.3)$$

这里的函数 f 是一个神经网络, 可以提取输入 box 的特征, 增强表达能力, 并用 sigmoid 函数归约到 $(0,1)$ 区间内。

10.4.3 Query2Box 详解

依然用 What are drugs that cause Short of Breath and treat diseases associated with protein ESR2? 这样的一个 query 作为例子, 我们可以先将这个 query 处理为: ((e:ESR2, (r:Assoc, r:TreatedBy)), (e:Short of Breath, (r:CausedBy))), 然后我们就可以使用投影, 从 head 实体开始一步步投影进行投影, 得到如下结果:

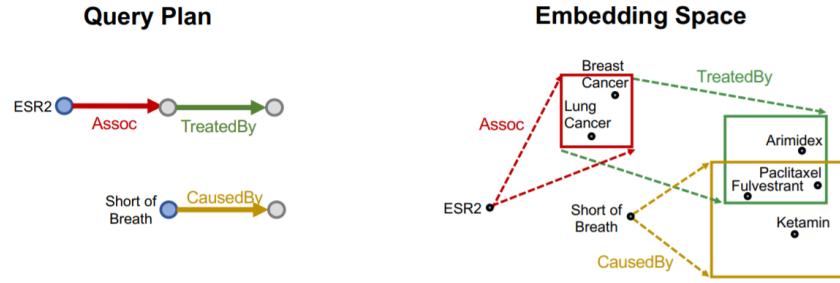


图 10.8: Query2Box 流程 1

然后在进行求交集的操作，得到最终的结果（阴影部分）

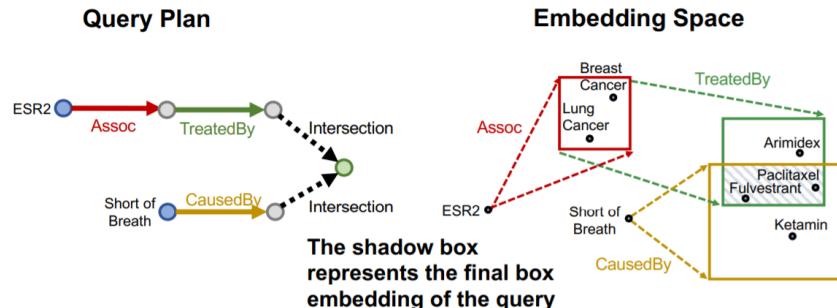


图 10.9: Query2Box 流程 2

10.4.4 打分函数

Query2Box 的知识图谱推理框架中，使用实体到 box 的负距离来定义打分函数，对于一个查询 q 和一个嵌入空间中的实体 v ，我们定义其距离函数为：

$$d_{\text{box}}(q, v) = d_{\text{out}}(q, v) + \alpha \cdot d_{\text{in}}(q, v) \quad (10.4)$$

这里的距离被分成了外部距离和内部距离两个部分，外部距离就是到 box 的边界的距离，而内部距离是 box 的边界到中心点的距离，而当一个实体在 box 内部的时候，其内部距离应该定义成负权重的，因此打分函数可以定义为：

$$f_q(v) = -d_{\text{box}}(q, v) \quad (10.5)$$

和知识图谱嵌入一样，我们希望这个打分函数对于正确的 (q, v) 而言分数更高，而对错误的分数更低，因此训练的时候需要用到负样本。

10.4.5 Union 操作的扩展

在 Query2Box 的框架下，交集的操作已经有了良好的定义，而并集操作是否也可以使用低维的向量来表示呢？一系列析取 (disjunction) 的连续查询又被叫做 Existential Positive First-order (EPFO) queries，这类查询可以分解为一系列 AND-OR 操作。

然而 AND-OR 查询不能用低维的向量来表示，比如有 3 个不同的 query 和对应的结果，可以使用 2 维空间表示，而如果有 4 个不同的 query，当下面这种情况出现的时候就需要 3 维的空间：

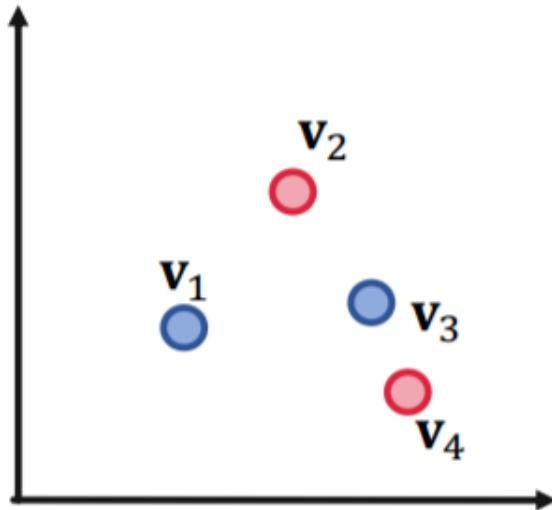


图 10.10: 一个反例

这是为了保证任意两个 query 的并集中不能包含其他的 query，而对于 M 个组合的 query，表示出所有的 OR 操作至少需要 $O(M)$ 规模的嵌入空间，因此数据规模一大就不能用低维向量来表示所有的并集操作。

但这个问题也不是完全没有解决的办法，论文中提出的一个 key idea 就是改变处理一个联合查询时候的顺序，到最后一步再做 union

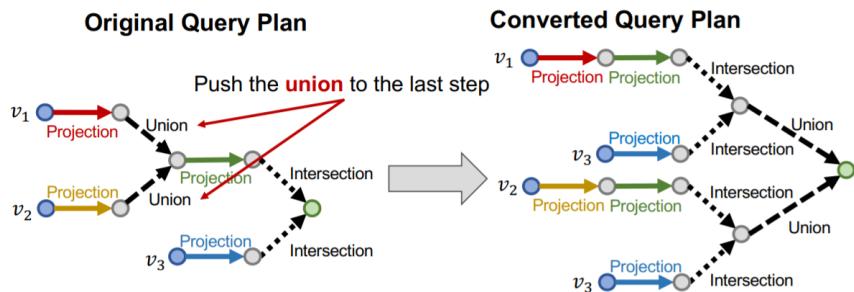


图 10.11: 查询优化

具体的做法是，任何一个 AND-OR 查询可以转变成一个等价的析取范式 (DNF)，即 $q = q_1 \cup q_2 \cup \dots \cup q_m$ ，这样一来我们来处理一个查询的时候就可以先求出所有 q_i 的嵌入，到最后一步再进行 union 操作。在最后计算距离作为 score 的时候，容易得到有：

$$d_{\text{box}}(\mathbf{q}, \mathbf{v}) = \min(d_{\text{box}}(\mathbf{q}_1, \mathbf{v}), \dots, d_{\text{box}}(\mathbf{q}_m, \mathbf{v})) \quad (10.6)$$

10.4.6 Query2Box 的训练

在 Query2Box 模型中，需要训练的参数主要有所有的实体的嵌入向量，所有的关系的嵌入向量和 Intersection 运算中的各种参数。

一个很直观的想法是，在训练 Query2Box 模型的过程中，对于一个查询 q 的嵌入向量，我们要让属于 q 中的实体 v 对应的打分函数最大化，而要让不在其中的打分函数最小化，为此需要用到负采样，也就是在训练的过程中，对于每个正样本 v 随机选取一个负样本与之对应，具体的训练过程可以分为以下几个步骤：

- 对输入的样本进行负采样
- 计算查询 q 的 box embedding

- 计算正负样本的打分函数 $f_q(v), f_q(v')$
- 计算损失函数并进行优化, 每个样本训练过程中的 loss 是 $\ell = -\log \sigma(f_q(v)) - \log(1 - \sigma(f_q(v')))$

10.4.7 query 的生成

在训练之前我们需要提取出一系列查询, 而这个过程称为 Query Generation, 可以通过一系列模板生成:

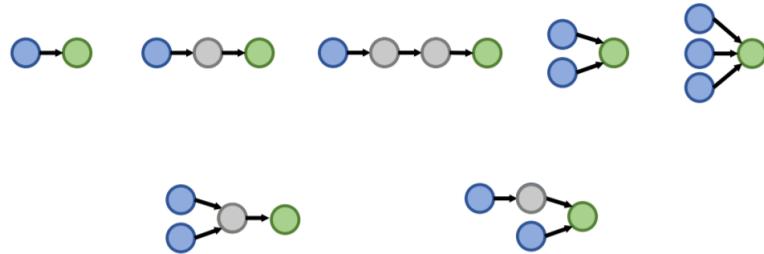


图 10.12: query 模板

第 11 章 频繁子图挖掘 Frequent Subgraph Mining

这一节内容主要介绍使用传统方法和 GNN 来进行频繁子图挖掘的内容。这就涉及到子图的概念，子图是一个图结构中的一些节点和这些节点之间的一些边构成的结构，是图的基本组成单元，并且可以表示一定的图特征，用于区分和判别不同的图结构。

11.1 子图

这里介绍了两种不同的子图，一种是 Node-induced subgraph 另一种是 Edge-induced subgraph，区别在于 Node-induced subgraph 构建的时候先取顶点的子集，然后从保留的顶点中选出一些边保留，而 Edge-induced subgraph 构建的时候先取边的子集然后保留其中的一些节点作为图的顶点集合。

构建子图究竟选用哪种方式还要取决于对应的领域，在化学领域的分子结构图中往往用 node-induced，因为化学中存在各种各样的官能团，而在知识图谱中用的比较多的是 edge-induced 对于给定的两个图 $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ ，如果存在一个定义在 $V_1 \rightarrow V_2$ 双射函数 f 满足对于 $(u, v) \in E_1 \rightarrow (f(u), f(v)) \in E_2$ ，那么就称这两个图是同构的 (Isomorphism)

11.1.1 图的同构

子图同构：如果图 G_1 和图 G_2 的子图是同构的，就称 G_2 关于 G_1 是子图同构的，这种情况下我们也可以 G1 是 G2 的子图。



图 11.1: 图的同构

11.1.2 网络主题 Network Motifs

网络主题 Network Motifs 是在一个图结构中的频繁出现且比较重要的模式，用子图的形式表现出来，Network Motifs 可以帮助我们理解图中存在的隐含信息并据此做出相应的决策，比如。常见的 Motifs 有：

- Feed-forward loops 前馈关系，常见于神经网络
- Parallel loops 代表了一种平行的关系，常见于食物网
- Single-input modules 代表了一种从属关系，常见于基因关系控制网络

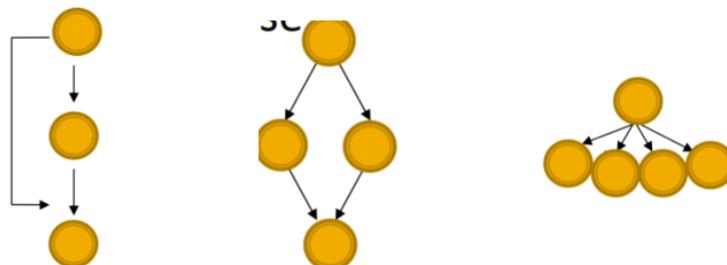


图 11.2: Motifs

11.1.3 子图的频率

对于一个图结构 G_Q 和目标图数据集 G_T , 节点 v 是 G_Q 中的一个点, 则节点 v 的子图频率并定义为:

- u 是 G_T 中的一个节点, G_T 中和 G_Q 同构并且将 u 映射到 v 的所有子图中, 所有节点 u 的个数
- (G_Q, v) 被定义成 node-anchored subgraph(节点固定子图)
- 如果图数据集中有多张图, 就分别进行计算

11.2 Motif 的重要性

一般来说相比于一个随机图, 在真实的图结构中出现频率越高的子图的重要性越高, 我们需要用一定的规则来定义子图的重要性。

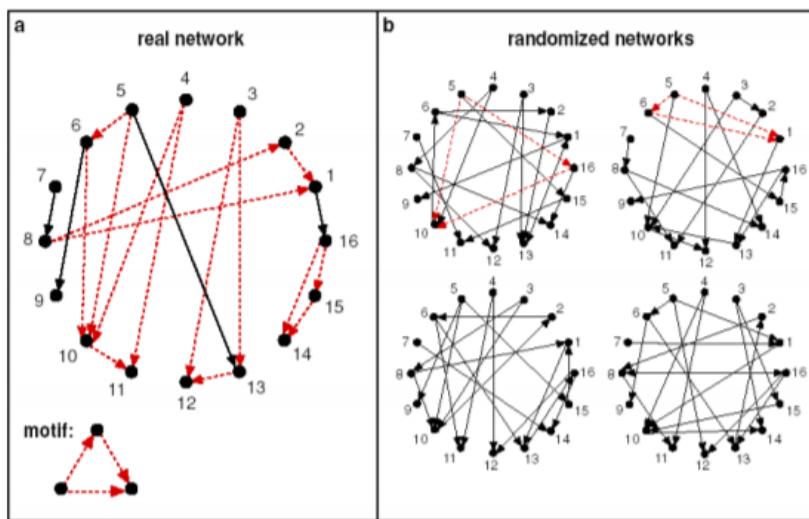


图 11.3: 真实图和随机图的对比

11.2.1 随机图

Erdős-Rényi (ER) random graphs 是由 n 个节点并且两两之间用概率 p 连接一条边的无向图, 记为 $G_{n,p}$

Configuration Model 可以生成一个每个节点的度数为指定值的图, 这种模型经常被用作图的 null-model, 我们可以通过比较真实的图 G 和一个随机生成并且每个节点的度数和 G 相同的随机图来分析比较得出一个图的重要性。这个模型生成随机图的过程如下:

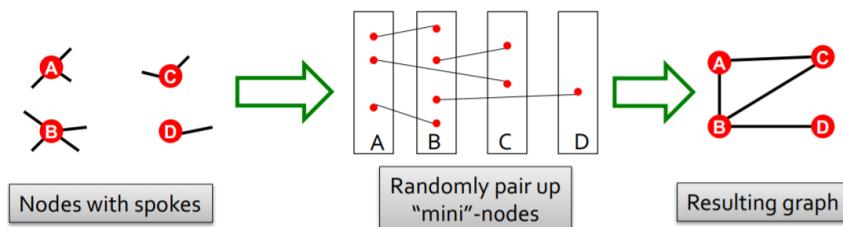


图 11.4: Configuration Model

另一种方法被称为交换 (Switching), 这种方法从一个给定的图开始, 重复若干次交换的操作, 每次选择两条边互相交换, 最后生成了一个随机的关联图, 这种图没有改变原本节点的度数, 只随即改变了边。

11.2.2 重要性计算的过程

一般来说 motif 在真实的图中出现的频率会比随机图中高很多 (overrepresented)，因此计算 motif 的重要性一般有如下几个步骤：

- 计算给定的图中的 motif 的个数
- 生成统计信息相近的随机图并且计算随机图中的 motif 数量
- 使用统计学评估方法来衡量每个 motif 对于这张图的重要程度

11.2.3 Z-Score

对于图中的一系列 motif，可以用 Z-Score 统计量来衡量其重要程度，对于第 i 个 motif，用 N_i^{real}, N_i^{rand} 分别表示在原图和随机图中发现的 motif 个数 (随机图可以有多个)，那么其 Z-Score 可以用如下方式计算：

$$Z_i = \frac{N_i^{real} - \bar{N}_i^{rand}}{\text{std}(N_i^{rand})} \quad (11.1)$$

并且可以对一系列的 Z-Score 进行标准化，得到网络重要性指标 significance profile(SP)

$$SP_i = \frac{Z_i}{\sqrt{\sum_{j=1}^n Z_j^2}} \quad (11.2)$$

SP 可以用来衡量不同的 motif 之间的相对重要程度

11.3 GNN 与子图匹配

子图匹配是一个很经典的图问题，需要判断一个给定的图是不是另一个图的子图，这一节的内容就是要用 GNN 来解决子图判别问题。一个很直观的想法是，我们可以通过比较嵌入空间的几何形状来提取子图同构的性质。然而一个令我不解的问题是，子图判断应该是一个确定性问题，为什么可以使用 GNN 来解决。

11.3.1 节点锚定

我们考虑将问题转化成一个二分类问题，即用 True 和 False 来判断一个图是否同构于另一个图的子图，我们可以考虑使用一种节点锚定 (node-anchored) 的方法：

- 我们可以用 GNN 去计算两个图中对应节点 u 和 v 的嵌入向量

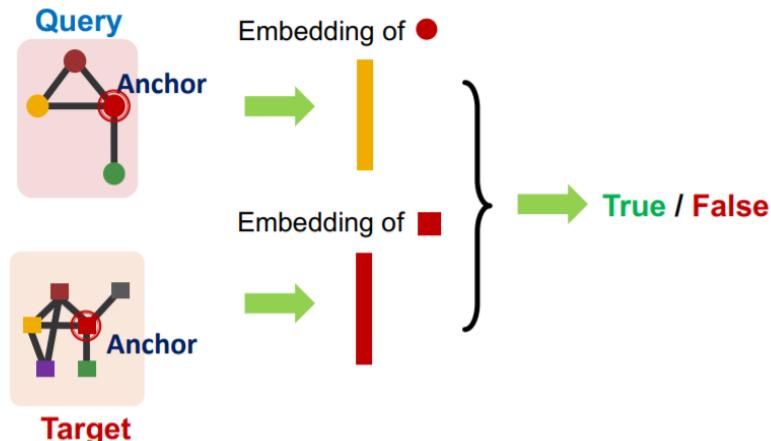


图 11.5：节点锚定 1

- 然后来判断 u 的邻居节点和 v 的邻居节点是不是同构的

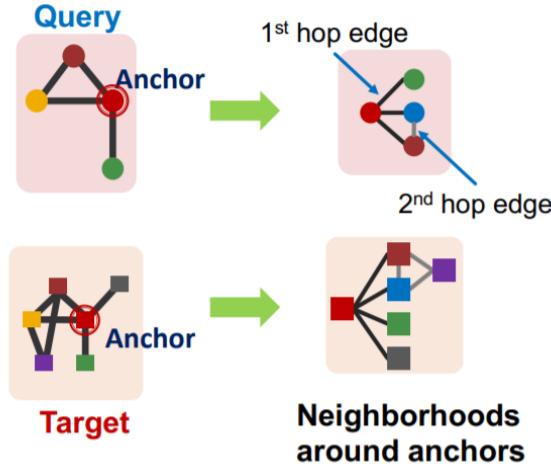


图 11.6: 节点锚定 2

这种方法不仅可以判断是不是子图，还可以同时标定一组对应的点，具体包括如下步骤：

- 假设我们需要在图 G_T 中匹配一个子图 G_Q
- 对于图中的每一个节点，计算出它的 K-Hop 邻居 (使用 BFS 等手段)，这里的深度 K 是一个超参数，可以自己设定
- 然后使用 GNN 得到邻居的嵌入向量

11.3.2 有序嵌入空间 Order Embedding Space

我们可以将图 A 映射到高维空间中的一个点，使得这个点的所有维度都是非负的，这样一些嵌入向量就可以比较大小，使用 \preceq 符号来表示不同嵌入之间的大小，一个直觉是子图往往在超图的 lower-left(可以理解为广义上的“左下角”)，对于一个子图和其超图，应该有：

$$\forall_{i=1}^D z_q[i] \leq z_u[i] \text{ iff } G_Q \subseteq G_T \quad (11.3)$$

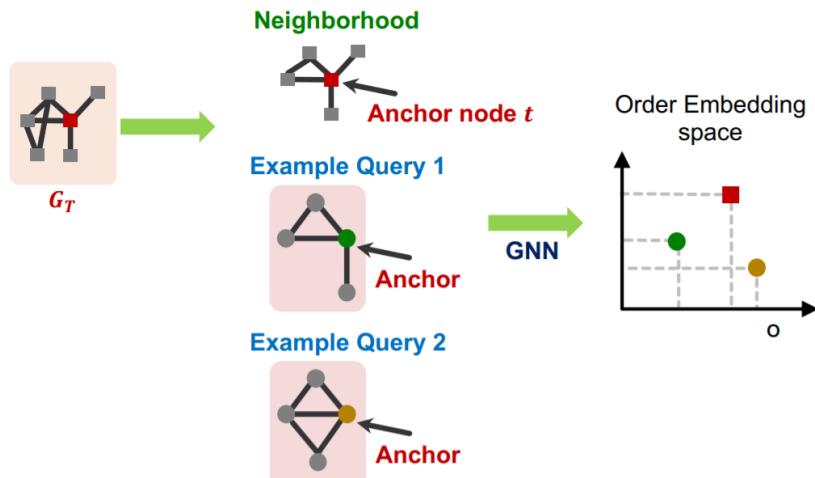


图 11.7: 有序空间嵌入 1

为什么可以使用有序嵌入？因为子图同构关系可以在有序嵌入空间中很好地编码，并且具有：

- 传递性， A 是 B 的子图， B 是 C 的子图，那么 A 是 C 的子图
- 反对称性，如果 AB 互为子图那么他们同构

- 闭包性，单个节点的图是所有非空图的子图，空节点是所有图的子图
- 这些性质都支持将一个图结构嵌入到一个有序嵌入空间中。

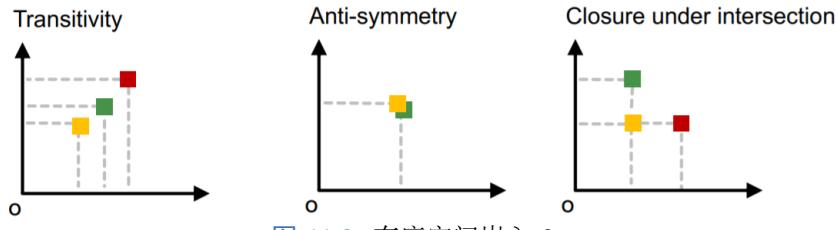


图 11.8: 有序空间嵌入 2

11.3.3 模型的构建和训练

我们已经说到用锚点和有序嵌入的方式来解决子图同构问题，为了构建并训练一个完整的模型，我们需要设定一个损失函数，来学习反映出子图关系的嵌入向量。

而损失函数的设定需要满足 Order constraint，这种约束可以保证子图关系的信息在嵌入空间中被保留了，因此可以设置损失函数为：

$$E(G_q, G_t) = \sum_{i=1}^D (\max(0, z_q[i] - z_t[i]))^2 \quad (11.4)$$

这个函数对于有子图关系的两个图来说就是 0，否则就是一个正数。我们可以采用负采样的方式来训练模型，对于样本 (G_q, G_t) 选取一半为有子图关系的正样本，而另一半为没有子图关系的负样本，并且使用一个 max-margin 损失函数，即：

- 对于正样本，最小化 $E(G_q, G_t)$
- 对于负样本，最小化 $\max(0, \alpha - E(G_q, G_t))$ ，这里的 α 是一个自己设定的 margin，我们希望负样本 (G_q, G_t) 计算出来的惩罚函数要尽可能小，使用这种形式的损失函数可以防止模型在训练的过程中，负样本 (G_q, G_t) 的嵌入越来越远离正确的位置
- 在训练的过程中每一步要采样一对正负样本进行训练，同时 BFS 的深度 K 可以作为超参数自己设置

11.3.4 频繁子图挖掘

频繁子图挖掘需要找到一个图中出现次数最多的大小为 k 的 motif，这需要解决两个问题，一个是枚举出所有大小为 k 的子图，另一个是统计不同类型的 motif 的个数。

然而频繁子图挖掘也是一个很困难的问题，困难的点在于频繁子图的寻找是一个 NP-hard 的问题（同构子图的判定是一个 NPC 问题），计算复杂度非常高，因此这一节课的内容提出了使用 GNN 来解决频繁子图挖掘的问题。

11.3.5 使用 GNN 进行频繁子图挖掘

GNN 或者说表示学习的方法可以帮助我们找到频繁子图，具体的解决方案是用控制搜索空间来解决组合爆炸的问题，然后用 GNN 来解决同构子图匹配的问题。我们可以将问题定义为如下形式：

- 假设需要操作的图是 G_T ，需要挖掘的子图的大小是 k ，并且得到的结果为 r
- 算法的目标是找到 G_T 中所有节点个数为 k 的子图并得到出现最高的频率，同时使用 node-level 来定义子图的概念

下面主要介绍一种频繁子图挖掘的 GNN 模型，其名为 SPMiner

11.3.6 SPMiner 算法

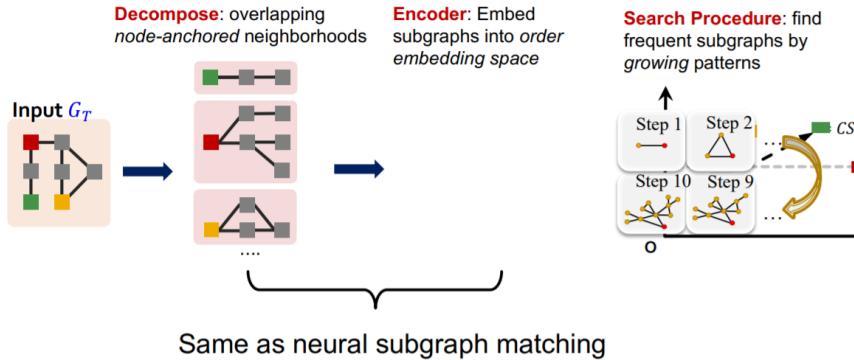


图 11.9: SPMiner

- SPMiner 将输入的图 G_T 分解成 neighbor，并计算其 order embedding，使用 order embedding 我们可以快速找到子图出现的频率
- 可以通过计算一个 order embedding “右上方”的 embedding 数来计算子图出现的频率，下图中红点右上方的所有点表示是 G_Q 的 neighbor 中出现过红点对应的子图

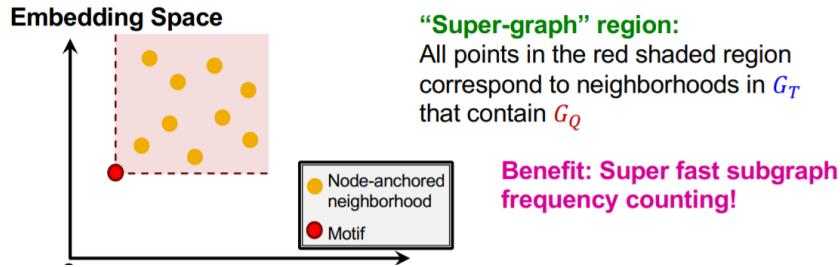


图 11.10: SPMiner-频率估计

SPMiner 算法的搜索过程包含以下几个步骤：

- 启动：从图中随机选择一个节点 u 开始搜索，定义集合 $S = \{u\}$ ，并按照上面提出的频率估计方法来统计

Walk in Embedding Space

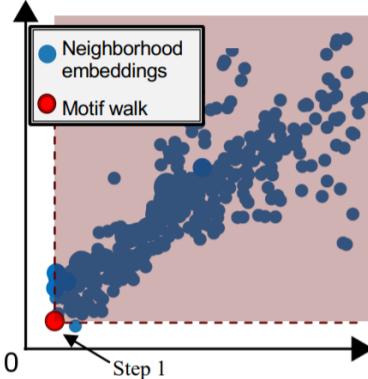


图 11.11: SPMiner-启动搜索

- 迭代步骤：从 S 的邻居中选择一个节点并将其加入 S ，同时重复上述搜索过程

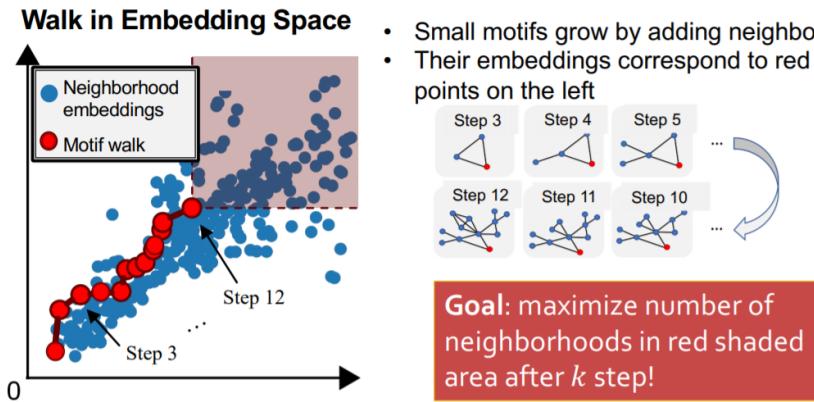


图 11.12: SPMiner-迭代步骤

- 当集合 S 达到了预设的 size 时，记录一个子图搜索结果

这里的邻居节点选择有一定的策略，Total violation of a subgraph G 指的是不包含子图 G 的邻居，将其最小化等价于将频率最大化，因此每一步搜索的过程中都采用贪心策略，将能带来最小化的 total violation 的节点加入集合 S

第 12 章 社区检测

我们通常会有这样一种概念，认为一个网络结构应该长成这样：

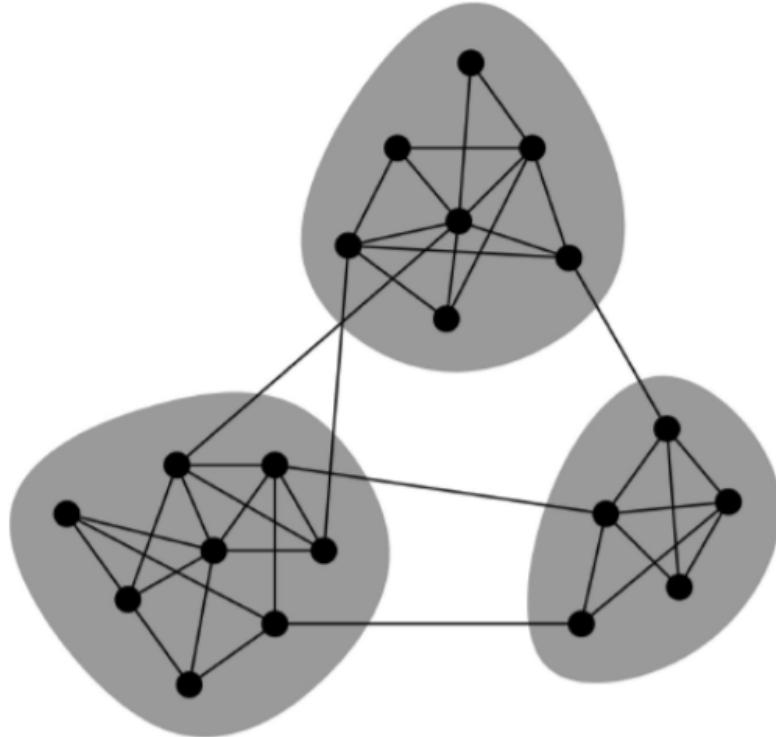


图 12.1：“社区”

我们发现这样的网络结构呈现出一种“大杂居小聚居”的形态，会有一些点组成小集群，同时边也可以分成 long 和 short 两种形式，比如我们用下面这样一个社交网络作为例子：

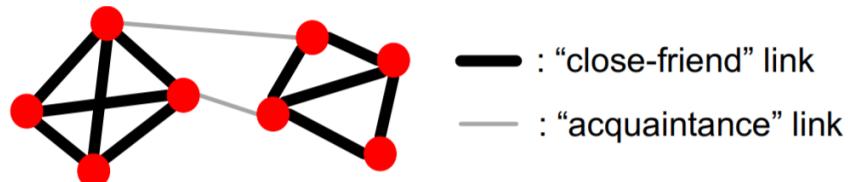


图 12.2：社交网络

- 在这个图结构里面，人依照关系亲密进行了划分，形成了一些团簇。粗线表示的是亲密朋友关系，而细线表示的是熟人关系。
- 一项研究发现在社交网络中，人们往往更喜欢去寻求熟人而不是朋友的帮助来找工作，这就说明在这个问题中细线边比粗线边更重要，在社交网络的信息传递中起到更重要的作用。
- 因此科学家对社交网络中的各种边进行了结构和社会学双重意义上的分析，将边分成了强弱两种

12.1 Edge Overlap

edge overlap 可以用来判断一条边是不是一个 local bridge(即连通了两个不相交的节点集群的边), 被定义为:

$$O_{ij} = \frac{|(N(i) \cap N(j)) - \{i, j\}|}{|(N(i) \cup N(j)) - \{i, j\}|} \quad (12.1)$$

如果一个 edge overlap 的计算结果是 0, 就说明这条边是 local bridge

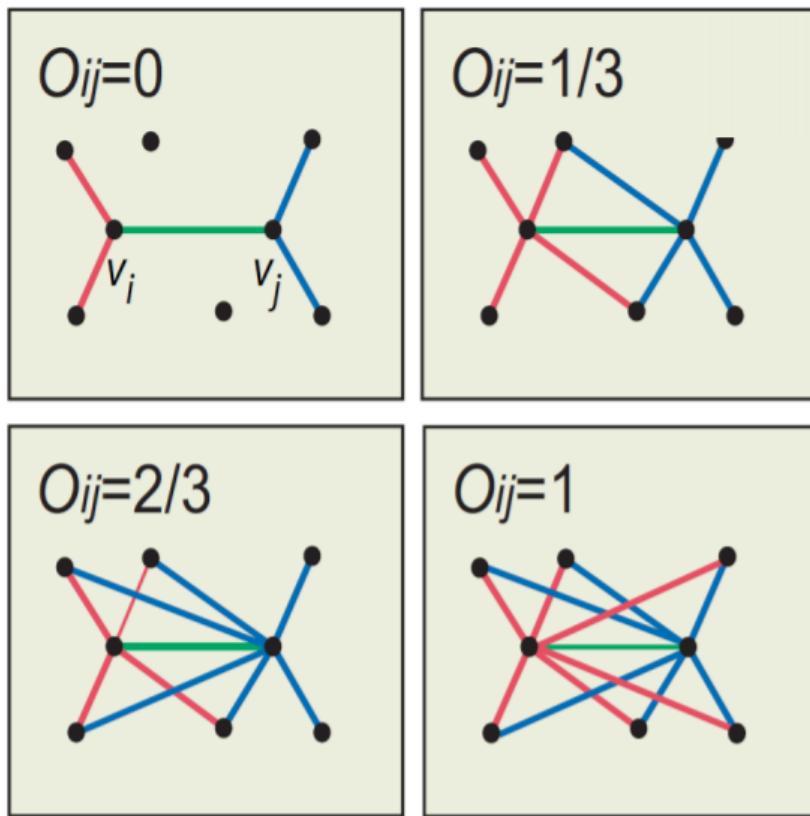


图 12.3: Edge Overlap

12.2 网络社区检测

网络社区 (Network Community) 被定义为是图结构中有较多内部连接和较少外部连接的点的集合, 我们可以根据网络社群对节点进行聚类。

12.2.1 Modularity 模块度

可以用模块性 Modularity 这个量来表示一个图是否被很好地被分成几个社群, 可以用来检测图分割的质量, 如果将图 G 分割成了若干个部分构成集合 S, 那么 Modularity 的计算方法可以表示为:

$$\sum_{s \in S} [\#e \in s - \text{expected}\#\#e \in s] \quad (12.2)$$

也就是说对于一个分割中的每一个 group, 计算其边的数量和应有的边的数量之差并求和。而这里需要一个 null model 来作为对照, 我们可以对一个有 n 个节点 m 条边的图 G 随机

生成一个度数分布与之相同的图，并且假设每个节点的度数为 k_i ，则有：

$$\sum_{i=1}^n k_i = 2m \quad (12.3)$$

两个节点 i 和 j 之间期望的边的条数是 $\frac{k_i k_j}{2m}$ 这种计算方式在加权图和不加权的图中都可以使用，这样一来就有了计算按照期望应有的边的数量的方式，modularity 的计算方式可以表示为：

$$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \in [-1, 1] \quad (12.4)$$

这个计算方式可以等价地表示成：

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \quad (12.5)$$

一般来说大于 0.3-0.7 就可以认为这是一个重要的社群结构，同时我们通过让 Q 最大化来寻找图中存在的社群。

12.3 社区检测算法

这一部分主要讲了两种社区检测算法，一种是 Louvain 算法，基于贪心策略，时间复杂度达到了 $O(N \log N)$ ，另一种是可以发现网络结构中的 overlapping community 的 BigCLAM 算法，具体的就不深入了解了。

第 13 章 图生成模型

我们希望通过图生成模型来生成现实世界中的图，通过这种方式可以深入了解图的内在形式，并且可以预测一张图未来的变化趋势，进行图的异常检测。

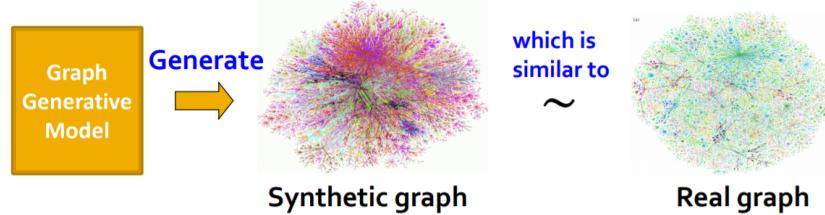


图 13.1: 图生成

13.1 传统的图生成模型

13.1.1 真实世界中图的性质

我们通常使用一些关键的特征来描述一张图，比如度数的分布，聚类系数，连通分量，路径长度等。这些前面都已经讲过了，这里就不再一一介绍。同时前面还讲了随机图的生成方法，其中比较有用的是 Erdős-Rényi 算法。

13.1.1.1 鲁棒性的衡量：Expansion

扩展性的定义是，对于图 $G = (V, E)$ ，对于任意的 V 的子集 S ，离开 S 的边的数量都大于 $\alpha \min(|S|, |V - S|)$ ，那么 α 就被称为是图 G 的扩展性，即：

$$\alpha = \min \frac{\#\text{(edge leaving } S\text{)}}{\min(|S|, |V - S|)} \quad (13.1)$$

- 一个 n 个节点的图中至少存在 $O(\log n / \alpha)$ 长度的路径
- 随机图因为有比较好的扩展性因此 BFS 所需的时间是 $\log N$ 级别的

13.1.1.2 随机图模型中存在的问题

随机图模型通过随机的方式生成一张图，并且依然可以保持比较好的路径平均长度，聚类系数，度数分布等特征，但是也存在如下问题：

- 缺少局部特征，聚类系数太低
- 度数的分布缺乏真实性
- 难以生成实际图中可能存在的大分量

13.1.2 Small-World 图模型

Small-World 图模型提供了一种使用常规的点阵图和随机图进行插值 (Interpolate) 的办法来生成高聚类系数并且低路径长度 (引申出一个概念叫做图的直径)

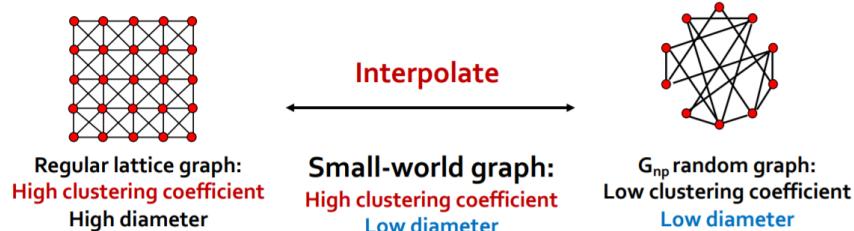


图 13.2: Small-World 图模型

首先从一个普通的点阵图开始，然后在这个图中引入一定的随机性，随机添加/删除一些边

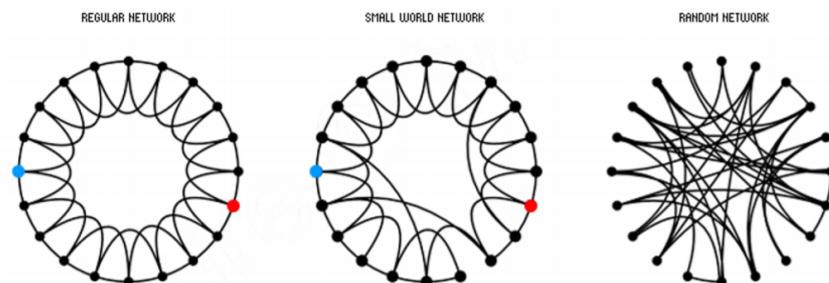


图 13.3: Small-World 图模型生成图

13.1.3 Knnonecker 模型

我们可以尝试使用递归的方法来生成图结构，而图结构的递归生成可以考虑自相似的方法，自相似指的就是整体和部分具有一定的相似性。而 Knnonecker 模型就提供了一种生成自相似模型的方法

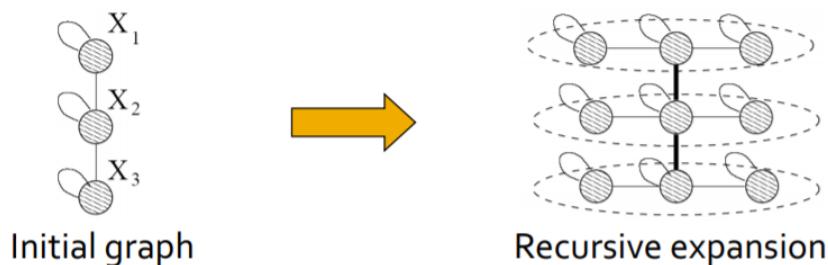


图 13.4: Knnonecker 模型

可以用两张图的邻接矩阵，定义其 Knnonecker 作为生成图的邻接矩阵，Knnonecker 积的计算方式如下：

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} \doteq \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{pmatrix}_{N^*K \times M^*L}$$

图 13.5: Knnonecker 积的计算

- 使用这种方法可以生成 m 阶的 Knnonecker 图，即 $K_m = K_{m-1} \otimes K_1$ ，在此基础上

13.2 深度图生成模型

13.2.1 机器学习方法

基于机器学习方法的图生成模型主要需要从给定的 $p_{data}(G)$ 中获得的数据分布，学习出数据的一个分布 $p_{model}(G)$ ，并且使用这个分布来生成所需要的图，我们用 $p_{data}(G, \theta)$ 来表示模型，我们希望得到一个尽可能接近 $p_{data}(G)$ 的结果，并且可以从这个模型中进行采样并生成结果。为此，我们可以使用极大似然法：

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{x \sim p_{data}} \log p_{model}(\mathbf{x} | \theta) \quad (13.2)$$

而图生成的过程中，我们可以先使用一个噪声分布 $z \sim N(0, 1)$ 来获取 $x_i = f(z_i; \theta)$ 这里的 f 就可以使用一个神经网络，而 x_i 被理解为是一系列动作，比如增加节点和增加边，因此一张图的生成可以用链式法则来表示：

$$p_{model}(\mathbf{x}; \theta) = \prod_{t=1}^n p_{model}(x_t | x_1, \dots, x_{t-1}; \theta) \quad (13.3)$$

13.2.2 GraphRNN

13.2.2.1 模型简介

GraphRNN 就是将图的构建看成了一种序列化的操作，即每次在图中添加节点或者在已有的节点中添加边（分别对应 node-level 和 edge-level 的任务），这样一来图的生成就变成了一个序列生成的问题，可以使用 RNN 来处理序列化的问题。

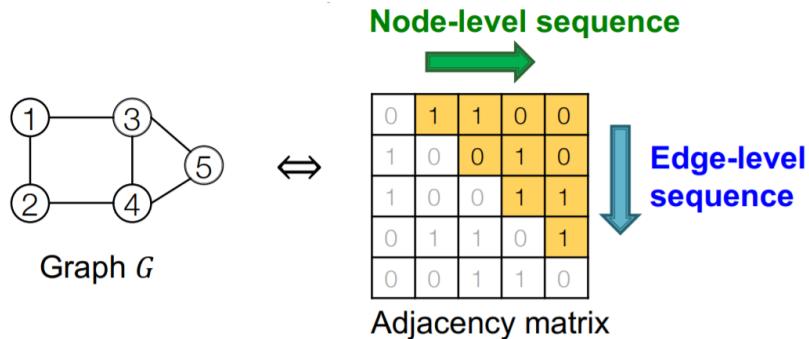


图 13.6: 图生成的序列化分解

我们需要解决两个问题：

- 生成新节点的状态，表示在序列中增加一个节点
- 根据节点的状态来生成对应的边

因此 GraphRNN 包含了两个 RNN，一个是节点级别的 RNN，另一个是边级别的 RNN，由 Node-RNN 来生成 Edge-RNN 的状态，并且由 Edge-RNN 来判断每一个新的节点是否会和前面的节点相连。

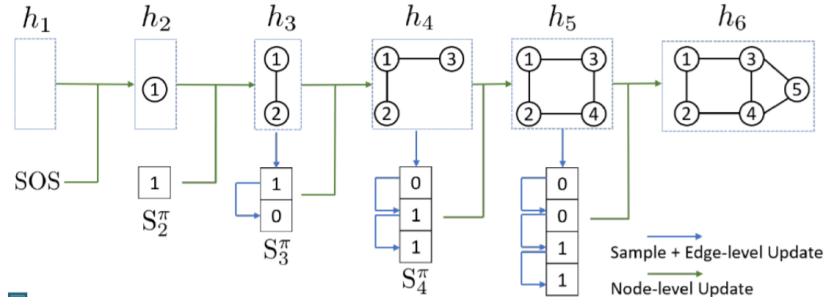


图 13.7: GraphRNN 工作原理 1

- 模型中使用 SOS 和 EOS 分别表示分别表示序列生成的开始和结束，同时我们可以将前一个的输出用作下一个的输入，来达到连续生成的目的
- 每一个步骤中 RNN 都会生成一个单条边的概率分布，然后从这个分布中进行采样，作为下一个步骤的输入

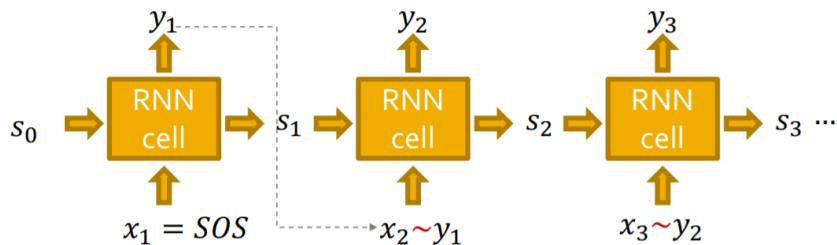


图 13.8: GraphRNN 工作原理 2

13.2.2.2 模型的训练和测试

在测试阶段，模型的运行方式如下，使用上一步的输出结果 y 生成下一个 x 的伯努利分布，然后进行采样得到 x 进行下一步。

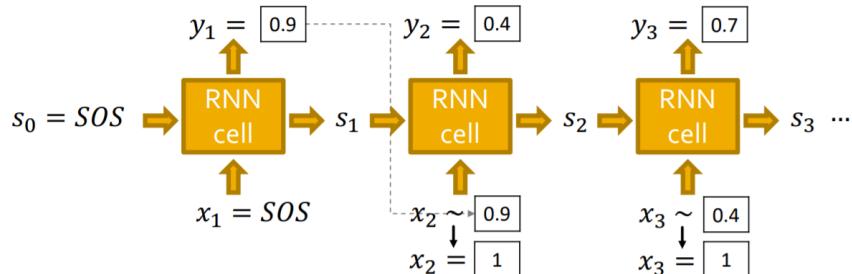


图 13.9: GraphRNN 测试阶段

而在训练的过程中还需要进行 Teacher Forcing，计算其损失函数并进行参数优化，这里损失函数一般使用交叉熵，即

$$L = -[y_i^* \log(y_i) + (1 - y_i^*) \log(1 - y_i)] \quad (13.4)$$

使用损失函数反向传播进行梯度优化可以使得预测结果 y_i 适应于输入数据的真实分布 y_i^* ，同时运行的过程中，一个图的生成可以分成以下几个步骤：

- 增加新结点：运行一个步骤的 Node-RNN 并且将其输出输入 Edge-RNN 进行初始化
- 给新节点添加新边：运行一次 Edge-RNN 并预测新的节点是不是会和前面的若干个节点相连
- 添加新节点：用最新的 Edge-RNN 的隐层作为输入运行一个步骤的 Node-RNN

- 停止生成：当 Edge-RNN 输出 EOS 的时候就停止生成

测试的时候和训练阶段最大的不同是，训练的时候要保留 Edge-RNN 获得的概率并将其用在 loss 的计算上，但是测试的时候直接按照概率分布来预测这条边存不存在并生成最后的结果。

13.2.2.3 存在的问题

图 RNN 生成一张图的过程比较复杂，对于要生成长路径的任务场景下表现不太好（这就好像 NLP 中的 RNN 会对长距离的依赖关系学习效果不佳一样），使用 BFS 的方法可以降低生成过程的复杂度。

13.2.3 评估生成的图

另一个很重要的问题是生成的图要怎么评估图生成模型生成的图的质量，我们可以用相似度的方式来定义生成图和真实图的相似性。而相似度的定义主要有两种方式，一种是 Visual Similarity，也就是使用眼睛来看，另一种方式是基于图统计信息的相似度，比如度数分布、聚类系数分布和轨道数等等。

Earth Mover Distance (EMD) 可以用来评估两个分布之间的相似性，计算方式是计算将一个分布“移动成”另一个分布所需要付出的代价，而 Maximum Mean Discrepancy (MMD) 通过将图划分成两个集合进行相似度的比较，可以用来评估 n 个图之间的相似性。

第 14 章 GNN 的前沿话题

本章内容继续回到图神经网络，来讨论一些 GNN 的高级话题，同时也是整个课程的结束部分。

14.1 GNN 的局限性

一个完美的 GNN 应该建立一个邻居结构到节点嵌入的单射，也就是一个邻居结构对应一个特定的单射，因此一个图中结构相同的节点的嵌入是一样的，结构不同的点的嵌入是不一样的。

而问题就在于，就算是两个邻居结构相同的点，我们有的时候可能也希望它们的嵌入表示不同，这类任务被称为是 Position-aware tasks，因此就算是一个完美的 GNN 也会在网格图等场景下失效。此外，基于消息传递机制的 GNN 不能学习出回路的长度，这就导致两个结构不同的节点学到的嵌入表示也可能是相同的（前面已经说过）

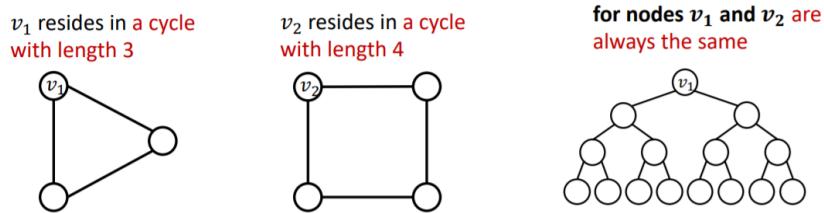


图 14.1: GNN 的局限性

解决这两个问题的办法分别是：

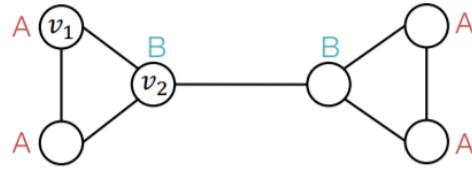
- 问题 1 的解决方法是基于节点的 position 来生成节点的嵌入，经典案例是:Position-aware GNNs
- 问题 2 的解决方法是使用更好的消息传递方式，比如 Identity-aware GNNs

对于这些问题，一个很 naive 的解决方式是使用 one-hot 编码，但是这种方式的可扩展性很差，因为嵌入的维度是 $O(N)$ 级别的。

14.2 Position-aware GNN

图中往往有两类任务，一类是基于结构的，一类是基于位置的，基于结构的任务需要考虑的关键是节点的结构，而基于位置的任务需要考虑的关键是节点在图中的“位置”，其区别就在于邻居结构是一种相对位置。GNN 往往在基于结构的任务中表现比较好而在基于位置的任务中表现不好。

Structure-aware task



Position-aware task

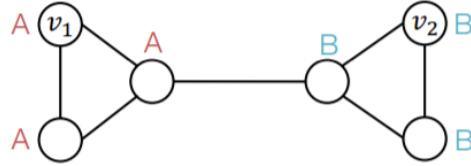


图 14.2: position-aware 的任务

一个常见的解决方法是选用锚点，然后用相对于锚点的距离来对点进行分类，在这个过程中锚点充当了一个坐标轴，当然锚点可以有多个，更多的锚点就好像使用了更多的坐标轴去刻画点的位置。同时可以用这种方法对图中的节点进行位置编码。

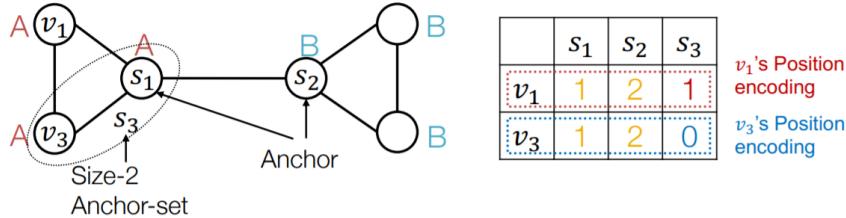


图 14.3: GNN 的局限性

要解决 Position-aware 的任务我们就必须使用跟 position 有关的信息，而是使用这些位置信息的方法有：

- Naive 的方法：使用位置编码作为额外的节点特征，值得注意的是位置编码向量的每一个维度是可以任意排序的，因为每个锚点的选取顺序不影响节点的位置特征
- Rigorous 的方法：使用一个神经网络来维护位置编码的任意排序中存在的不变信息

14.3 Identity-aware GNN

我们已经知道 GNN 在 Position-aware 的任务中表现不好，那么 GNN 在 Structure-aware 的任务中表现如何呢？事实上表现依然不好，在多个层级（节点，边和图）的 Structure-aware 中依然表现不好，不同的输入仍然可能导致相同的计算图，这就无法区分不同的 Structure

一种解决办法是，我们可以给每一个需要 embedding 的节点标上一个颜色，这样生成的计算图中也会有带颜色的节点，就可以区分出不同的节点，并且这种颜色不会随着节点顺序的变化而变化。

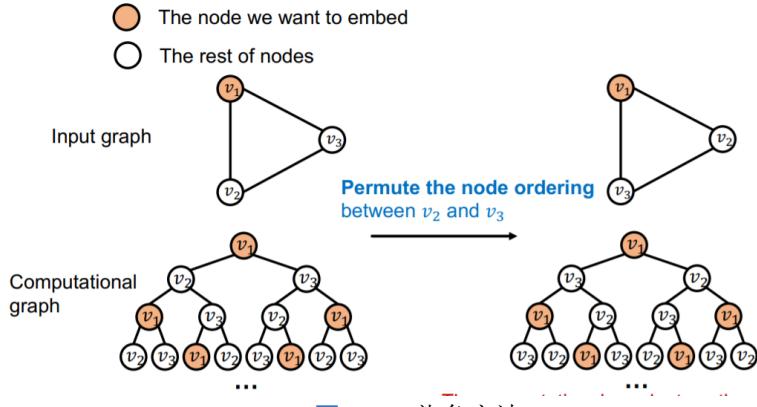


图 14.4: 染色方法

有了颜色之后，我们可以用颜色来区分计算图，这样一来之前会 failure 的例子也可以 work 了，这种方法在图分类任务中也可以使用。

而 Identity-aware GNN 正是使用了这种染色机制，并且使用了 ** 异构的消息传递 **，不同的节点之间的消息传递方式不同。一个 ID-GNN 中不同颜色之间消息传递的方式不同。因此 ID-GNN 可以挖掘出图中的环的信息，可以通过计算图来计算各种长度的环的个数。

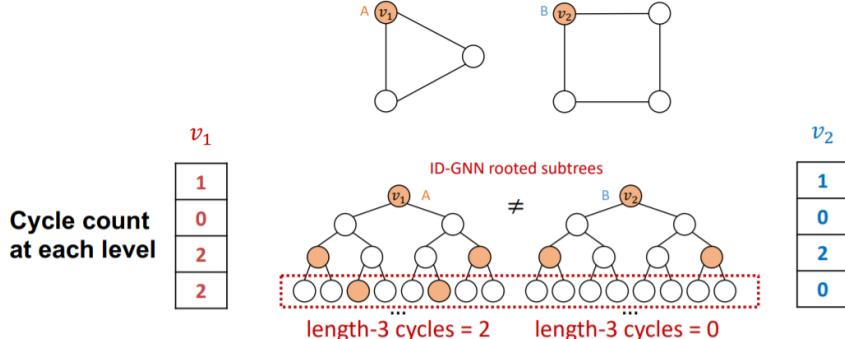


图 14.5: 染色方法

ID-GNN 的一个简化版本是 ID-GNN-Fast，这种模型会统计计算图中各种长度的环的个数，并将其作为一个额外的节点特征，而 ID-GNN 实际上是一个“插件”，可以被添加到任意一种基于消息传递机制的 GNN 中，并被实验证明是非常有效的。

14.4 GNN 的鲁棒性

神经网络已经在很多领域取得了一定的应用，但一个很现实的问题是，这些模型真的能被用到现实生活中吗？事实上神经网络容易遭到对抗攻击，比如 CNN

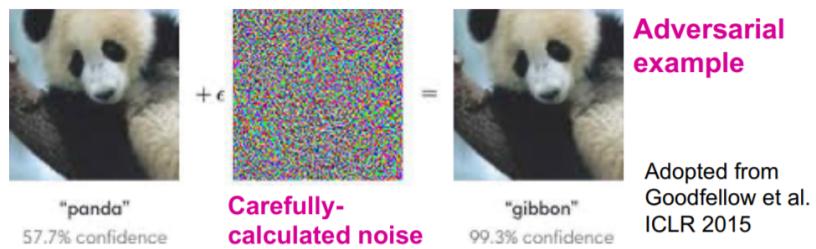


图 14.6: 深度学习模型的攻击

深度模型往往缺少鲁棒性，很容易被一些伪造的样本攻击导致性能极度下降，那么图神经网络的鲁棒性究竟如何呢？为了研究这个问题，我们假定问题是半监督的节点分类（只有少量的），而模型是 GCN

14.4.1 问题的描述

我们可以将节点分为目标节点 t 和可攻击节点集合 S ，攻击的方式可以分为直接攻击和间接攻击。

- 直接攻击是指，可以攻击的节点就是目标节点的情况。间接攻击是可以攻击的节点不是目标节点的情况
- 攻击的方式主要有两种，可以直接给目标节点改变特征，也可以增减攻击节点的边

14.4.2 问题的形式化

我们可以用形式化的数学语言来描述 GNN 对抗攻击这个问题，我们的目标是在尽可能少地改变图结构的情况下，尽可能改变目标节点预测的结果，这是因为如果把图改动的太离谱了，很容易就会被发现这是伪造的数据，因此要尽可能减少对图的修改。

我们假设图的邻接矩阵是 A ，图节点的特征矩阵是 X ，改变后的邻接矩阵和特征矩阵分别是 A' , X' ，且要求 $(A', X') \approx (A, X)$ ，并且假设需要预测的节点是 v ，从原始的图中 GCN 已经学习到了模型参数为：

$$\theta^* = \operatorname{argmin}_{\theta} \mathcal{L}_{\text{train}}(\theta; A, X) \quad (14.1)$$

原始的 label 预测结果为：

$$c_v^* = \operatorname{argmax}_c f_{\theta^*}(A, X)_{v,c} \quad (14.2)$$

而 GCN 在新的伪造图上学习到的模型为：

$$\theta^{*\prime} = \operatorname{argmin}_{\theta} \mathcal{L}_{\text{train}}(\theta; A', X') \quad (14.3)$$

而新图上节点 v 的预测结果是

$$C_v^{*\prime} = \operatorname{argmax}_c f_{\theta^{*\prime}}(A', X')_{v,c} \quad (14.4)$$

我们的目标是 $C_v \neq C_v^*$ ，为了让两个结果相差的尽可能大，我们可以把目标函数定义为：

$$\Delta(v, A', X') = \log f_{\theta^{*\prime}}(A', X')_{v,c_v^{*\prime}} - \log f_{\theta^*}(A, X)_{v,c_v^*} \quad (14.5)$$

接下来就需要针对这个目标进行优化，然而邻接矩阵 A' 是一个离散的对象，不能使用基于梯度的优化方法，因此我们需要采用一定的优化策略选择好参数 A', X' 之后重新训练一个 GCN 来验证结果的好坏，现有的方法都是对这个优化目标的近似解，具体的这门课上也没有深入讲下去，需要阅读相关论文来深入学习。Slides 里面举了一篇 KDD18 的论文作为例子，这篇论文中采用了直接攻击，间接攻击和不攻击三种策略，并在直接攻击中使用了随机攻击作为对照，得到的实验结果是：

14.4.3 实验结果和结论

Slides 里面举了一篇 KDD18 的论文作为例子，这篇论文中采用了直接攻击，间接攻击和不攻击三种策略，并在直接攻击中使用了随机攻击作为对照，得到的实验结果是：

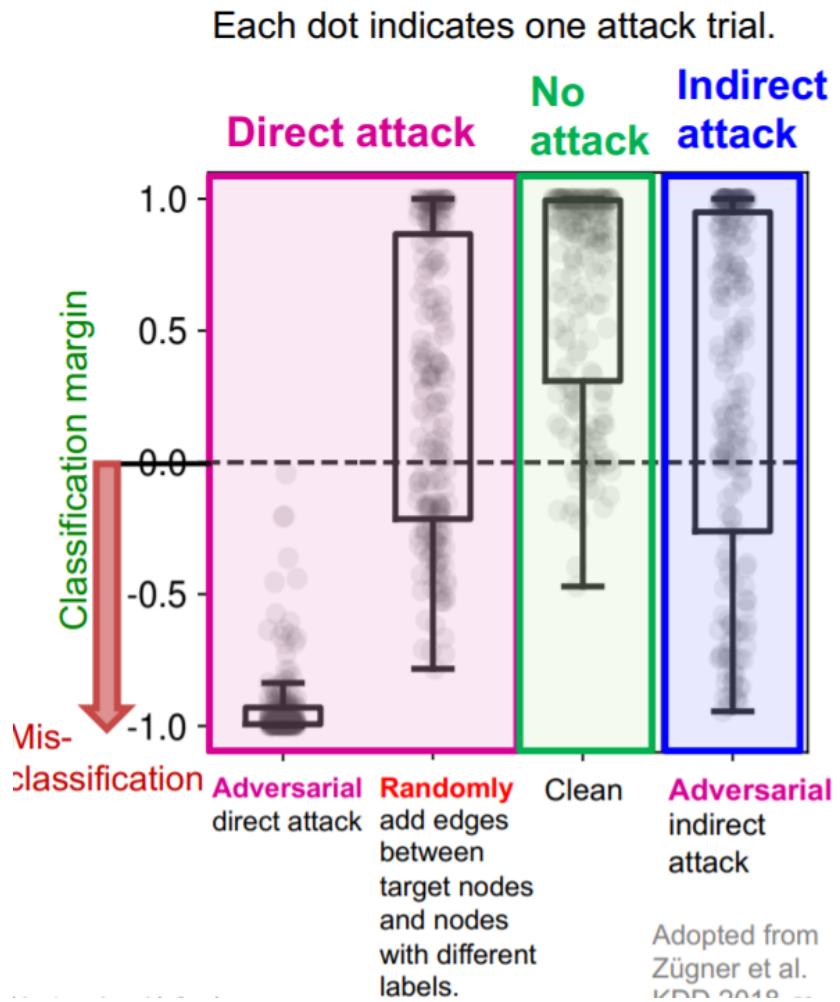


图 14.7: 鲁棒性实验结果

我们发现 GCN 对抗直接攻击的能力比较差 (对直接对抗攻击不鲁棒)，但是可以应付间接攻击和随机攻击。